VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

URČOVÁNÍ POLOHY ROBOTA NA ZÁKLADĚ MĚŘENÍ ZE SENZORŮ

ROBOT POSITIONING BASED ON SENSOR MEASUREMENTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Ondrej Čakloš
AUTHOR

VEDOUCÍ PRÁCE — Ing. Jaroslav Rozman, Ph.D.
SUPERVISOR

BRNO 2018

Abstrakt

Cieľom tejto práce je vytvoriť program, ktorý bude prijímať merania zo senzorov robota, previesť fúziu senzorov a vyhodnotiť polohu robota na základe tejto fúzie.

Pri riešení som použil znalosti o pravdepodobnostnej robotike, robotickom operačnom systéme, fúzií informácií, filtrovaní predovšetkým Kalmanov filter a lokalizácií robota.

Ako riešenie vznikla aplikácia rozšíreného Kalmanovho filtra. Filter naslúcha správam od senzorov, vytvára ich fúziu a následne vypočítava odhad polohy robota v priestore. Filter dokáže prijímať merania z viacerých zdrojov.

Výsledné hodnoty stavov sa preukázali ako dostatočne presné pre úspešné lokalizovanie robota v priestore.

Abstract

The goal of this thesis is to create a program, that will be receiving measurements from robot's sensors, provide sensor fusion and estimate position of robot based on this fusion.

For solving I used knowledge about probabilistic robotics, robotic operating system, information fusion, filtering especially extended Kalman filter and robot localization.

I created an application of extended Kalman filter as a result. Filter listen to messages from robot sensors, providing a sensor fusion and estimating position of the robot in environment. Filter can receive measurements from multiple sources.

The estimated states have proven themselves reasonably accurate for successful robot localization in space.

Klíčová slova

Fúzia senzorov, Global Positioning System, Kalmanov filter, Rozšírený Kalmanov filter, Lokalizácia robota, Navigačný zásobník, Odometria, Pravdepodobnostná robotika, ROS

Keywords

Senzor fusion, Global Positioning System, Kalman filter, Extended Kalman filter, Robot localization, Navigation stack, Odometry, Probabilistic robotics, ROS

Citace

Čakloš Ondrej: Určování polohy robota na základě měření ze senzorů, bakalářská práce, Brno, FIT VUT v Brně, 2018

Určovanie polohy robota na základe meraní zo senzorov

Prohlášení

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Jaroslava Rozmana, Ph.D. a s použitím uvedenej literatúry a prameňov.

……
Ondrej Čakloš
Datum (17.5.2018)

Poděkování

Rád by som poďakoval svojmu vedúcemu bakalárskej práce Ing. Jaroslavovi Rozmanovi, Ph.D. za odborné vedenie, za pomoc a rady pri spracovaní tejto práce.

Obsah

1 Úvod

Táto práca sa zaoberá fúziou informácií zo senzorov i ich následné spracovanie pre určenie polohy robota. Senzory ktoré roboti používajú sú rôzne a snímajú prostredie v ktorom sa robot pohybuje rôzne. Každý model robota je iný môže mať iný tvar, iné senzory alebo len iné nastavenia. Všetky tieto aspekty treba brať v úvahu pre čo najpresnejšie určenie polohy. Fyzické rozloženie robota a jeho komponentov je tiež dôležité pre jeho lokalizáciu a navigáciu.

Snímanie senzorov je spracovávane vnútorným programom robota, ktorý preposiela tieto informácie na sieť. Väčšina robotov pracuje na Robotickom Operačnom Systéme (skr. ROS). Ten zabezpečuje jak fungovanie robota tak aj komunikáciu s ním.

Pri určovaní polohy robota v priestore sa používa veľa druhov senzorov. Používané senzory sa menia od robota k robotovi. Od jednoduchých enkodérov alebo sonarov až po laserové merače vzdialenosti skoro v 360° okolí zvaný Lidar či rôzne verzie kinectu ktoré vytvárajú mračno bodov, kde poloha každého bodu je určená v troch súradných osách. Zistili sme že medzi najpoužívanejšie patria enkodéry, ktoré poskytujú odometriu, IMU, poskytujúce hneď niekoľko senzorov pohromade, globálny poziční systém GPS a už spomínaný lidar alebo kinect.

Fúziu meraní zo senzorov spracuje filter. Fúzia senzorov napomáha rýchlemu spracovaniu čo najväčšieho množstva senzorov súčasne. Z informácií získaných zo senzorov filter vypočíta odhad pozície robota. Pri výpočte musíme dbať na nepredvídateľnosť prostredia. Filtrov je viacero druhov a každý z nich má iné výhody.

Problém pri určovaní polohy robota v prostredí je v nepresnosti nameraných hodnôt zo senzorov. Z dôvodu týchto nepresností a predpokladu, že lokalizačný systém s týmito nepresnosťami počíta, môže sa niekedy javiť ako by sa robot nachádzal na viacerých miestach súčasne. Preto musíme zabezpečiť čo najväčšiu presnosť lokalizácie.

2 Robotický operačný systém

Ako viac než dostatočným zdrojom informácií ktorý som používal v tejto práci je dokumentácia Robotického operačného systému. [11]

Robotický operačný systém (ďalej len ROS) je meta-operačný systém s otvoreným zdrojovým kódom. Systém obsahuje knižnice a nástroje pre uľahčenie práce pri vytváraní softwaru pre rôznych robotov. Poskytuje implementácie často používaných funkcií, ovládanie nízkoúrovňových zariadení, hardwarovú abstrakciu, management **balíčkov (ang. package)** a posielanie správ medzi procesmi. Taktiež ponúka nástroje a knižnice pre získavanie, písanie, prekladanie a spúšťanie kódu na viacerých počítačoch.

ROS využíva niekoľko spôsobov komunikácie ako synchrónne RPC komunikácie cez **služby (ang. services)**, asynchrónny spôsob komunikácie cez **témy (ang. topic)**, a ukladanie dát na parametrový server. [10]

Aj keď kostra ROS-u nie je stavaná pre prácu v reálnom čase, je možné do ROS-u integrovať kód s reálnym časom.

2.1 Myšlienka ROS-u

ROS podporuje znovu-použiteľnosť kódu vo vývoji a výskume robotiky. ROS využíva distribuovanú štruktúru procesov tzv. **uzly (ang. node).** Umožňuje to individuálne tvorenie častí a voľné spájanie uzlov pri spúšťaní. Takto môžu byť procesy zhlukované do balíčkov a **zásobníkov (ang. stack)**, ktoré môžu byť jednoducho zdieľané.

Myšlienka ROS-u je zdieľanie a spolupráca. Okrem toho však má aj ďalšie ciele:

- Prostý kód – ROS je navrhnutý tak aby bol čo najprostejši jak je možné a tým pádom jednoduchý na integrovanie pre rôzne použitie.
- Knižnice nezávislé na ROS-e – je preferovaný model vývoja, písať nezávislé knižnice s jasnou funkcionalitou.
- Jazyková nezávislosť – štruktúra ROS-u je jednoducho implementovateľná na hociktorom modernom programovacom jazyku.
- Jednoduchosť testovania – ROS má vstavanú štruktúru na testovanie.
- Škálovanie – ROS je vhodný pre veľké systémy a vývojové procesy.

2.1.1 Podporované programovacie jazyky

Jazyková nezávislosť ROS-u spočíva v možnosti jednoduchej implementácie v ľubovoľnom programovacom jazyku. Aktuálne už je implementovaný v týchto troch programovacích jazykoch:

- **Jazyk C++** je univerzálny objektovo orientovaný programovací (OOP) jazyk, ktorý vytvoril Bjarne Stroustrup a je rozšírením pre programovací jazyk C. Preto je možné v C++ písať kód „C štýlom“, alebo „objektovo orientovaným štýlom“. Najväčšou výhodou jazyka sú preddefinované triedy. Tieto triedy sú dátové typy, ktoré môžu mať viacnásobné inštancie. Do tried môžu byť implementované funkcie pre zabezpečenie určitej funkcionality. Viacero objektov určitej triedy môže byť definované pre implementáciu funkcií vo vnútri triedy. Triedy môžu byť zdedené po iných, aj nových triedach, kde z pôvodnej triedy si v základe vezmú chránené a verejné funkcionality. Dôležité koncepty v C++ zahŕňajú polymorfizmus, predlohy (angl. templates), virtuálne

a priateľské funkcie (angl. virtual and friend functions), ukazatele (angl. pointers), menné priestory (angl. namespaces).

- **Python** je viac-paradigmový, univerzálny, interpretovaný, vysoko úrovňový programovací jazyk. Umožňuje programátorovi použiť rôzne programovacie štýly pre vytvorenie jednoduchých alebo komplexných programov, dostať rýchlejšie výsledky a písať kód, skoro ako bežný hovorený jazyk. Výhody sú, že python je interpretovaný jazyk, takže jeho kód nie je potrebné kompilovať pred spustením, a kód v pythone býva oproti ostatným jazykom kratší. Nevýhodou je jeho pomalšie spúšťanie kódu (v dnešnej dobe je už zanedbateľné).
- **Lisp** je vysoko úrovňový programovací jazyk. Časť jeho hlavných dátových štruktúr sú viazané zoznamy a jeho zdrojový kód je tvorený zoznamami. Pôvodne používaný ako praktická matematická notácia pre počítačové programy. Neskôr pre vykonávanie výskumov ohľadom umelej inteligencie.

Vývojári ROS-u sa stále snažia rozšíriť počet podporovaných programovacích jazykov. Momentálne skúšajú knižnice na jazyky Java a Lua. Pre svoju prácu som si zvolil programovací jazyk C++ pre jeho dostupnosť a univerzálnosť.[12]

2.2 Koncepcia

ROS rozlišuje tri úrovne koncepcie: Úroveň súborového systému, úroveň výpočtového grafu a komunitnú úroveň.

2.2.1 Úroveň súborového systému

Rozlišuje hlavne zdroje ROS-u na disku. Medzi ktoré patria balíčky, metabalíčky, manifest balíčku, repozitáre a typy správ a služieb.

Balíčky sú základnou jednotkou pre organizovanie softwaru v ROS. Môže obsahovať uzly, knižnice závislé na ROS, datasety, konfiguračné súbory alebo čokoľvek iné, kde všetko je užitočne zabalené dohromady. Balíček je to najatomickejšie, čo ROS dokáže zostaviť.

Metabalíček je špecializovaný balíček, ktorý slúži iba na označenie skupiny iných balíčkov.

Manifest balíčku poskytuje metadata o balíčku vrátane jeho mena, verzie, popisu, informáciách o licencii, závislosti a ostatné meta informácie.

Repozitár je kolekcia balíčkov, ktoré zdieľajú VCS (systém kontroly verzie). Balíčky, ktoré zdieľajú VCS, zdieľajú aj verziu a môžu byť vydané spoločne pomocou automatického nástroja catkin release nazývaný bloom.

Typy správ a služieb sú popisy, ktoré definujú dátové štruktúry pre správy resp. štruktúry požiadaviek a odpovedí pre služby. [8]

2.2.2 Úroveň výpočtového grafu

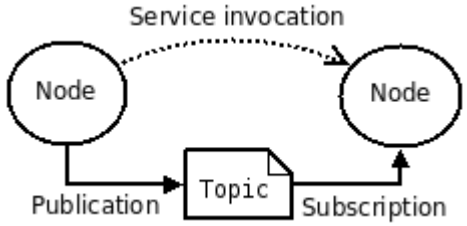
Výpočtový graf je peer-to-peer sieť ROS procesov a spracovávaných dát. Základné koncepty grafu sú: uzly, Master, parametrový server, správy, služby, témy a **vaky (ang. bags)**. Každý poskytuje dáta grafu rôznymi spôsobmi. [10]

- **Uzly,** sú procesy, ktoré vykonávajú výpočty. ROS je designovaný pre modularitu vo vysokej zrnitosti. Ovládací systém sa často skladá z veľkého počtu uzlov, kde každý uzol zastáva prevažne jedinú funkciu, ako ovládanie motorov kolies alebo lokalizáciu v priestore atď..

- **Master** poskytuje registráciu názvov a ich vyhľadávanie pre zbytok výpočtového grafu. Bez Master-u by uzly neboli schopné nájsť jeden druhý, vymieňať si správy, alebo zavolať služby.
- **Parametrový server** zase umožňuje ukladanie dát podľa kľúča v centralizovanej databáze. Momentálne je súčasťou *Masteru*.
- **Správy (ang. messages),** sú spôsob akým komunikujú uzly medzi sebou. Zjednodušene, sú to dátové štruktúry, ktoré obsahujú textové polia. Štandardné dátové typy, ako integer, sú podporované. Tak ako aj **array-e (zoskupenia)**. Správy môžu tiež obsahovať aj ľubovoľne vnorené štruktúry a array-e. Správy sú prepravované pomocou transportného systému *publish/subscribe,* kde uzol odošle správu pomocou *publishingu* na konkrétnu tému. Každá téma má pomenovanie, ktorým podáva informáciu o správach zasielaných na ňu. Uzol pomocou subsrcibu preberá správy z konkrétnej témy.. Na ktorúkoľvek tému môže odosielať, alebo z nej prijímať správy ľubovoľný počet uzlov. Taktiež jeden uzol môže odosielať a prijímať správy z ľubovoľného počtu tém.
- **Služby** na rozdiel od *publish/subscribe* modelu využívajú jednosmerný spôsob zasielania, žiadosť/odpoveď, ktoré sú často požadované pri distribuovanom systéme. Správy pre služby sú definované párom štruktúr, kde jedna štruktúra je pre žiadosť a druhá pre odpoveď. Takže ak uzol poskytuje službu, môže ju klient využiť pomocou zaslania správy s žiadosťou a následne si počkať na odpoveď.
- **Vaky** sú úložiskom pre ROS správy a senzorové dáta. Takto uložené správy je možné jednoducho znovu prehrať. To zjednodušuje proces vývoja a testovania algoritmov.

ROS Master slúži v výpočtovom grafe ako menný server. Ukladá si registračné údaje tém a služieb aby ich mali uzly ROS-u k dispozícií. Uzly komunikujú s Masterom aby oznámili registračné údaje. Tieto údaje potom môže Master poskytnúť ostatným uzlom pre zahájenie komunikácie. Pri akejkoľvek zmene informácií, Master dynamicky reaguje na túto zmenu, čím zjednodušuje a zrýchľuje komunikáciu pri spustení nových uzlov. Master však poskytuje iba vyhľadávacie údaje, uzly samotné nadväzujú spojenie medzi sebou. Uzly ktoré sa prihlásia na odber správ z témy vyžadujú spojenie s uzlami, ktoré vysielajú na danú tému a vytvoria spojenie na základe dohodnutom spojovacom protokole. Najpoužívanejší je TCPROS protokol, ktorý využíva štandardný TCP/IP sockety.

Takáto architektúra umožňuje oddelenú prevádzku, kde mená sú primárnym spôsobom akým môžu byť väčšie a komplexnejšie systémy postavené. Mená majú v ROS-e veľkú rolu: uzly, témy, služby a parametre to všetko má v ROS-e mená.



Obrázok 2.1 Zjednodušená schéma komunikácie

2.2.3 Komunitná úroveň

Komunitná úroveň zakladá na schopnosti výmeny softwaru a skúseností medzi rôznymi komunitami. K tomu slúžia napríklad:

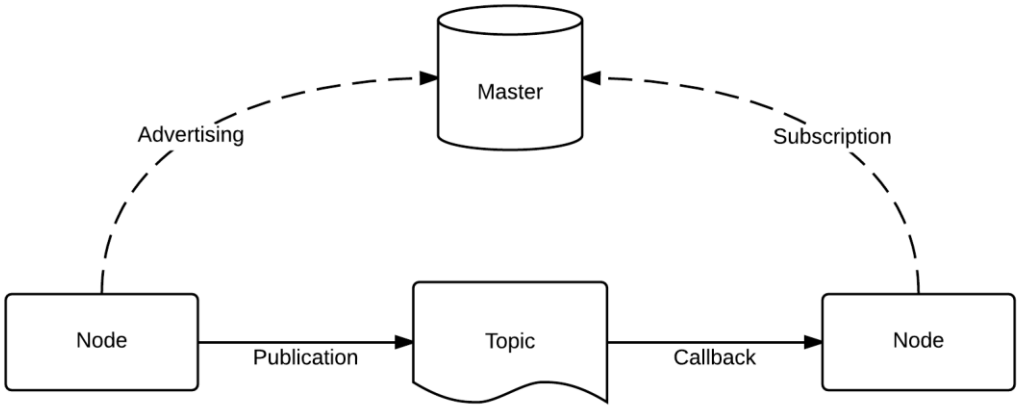
- Distribúcie – kolekcie verzií softwaru s možnosťou inštalácie.
- Repozitáre – ROS zakladá na spoločnej sieti repozitáru kódov, kde rôzne inštitúcie môžu vydávať ich vlastné softwarové komponenty.
- ROS wiki – je hlavným fórom pre dokumentácie pre ROS software.

2.3 Komunikácia v ROS-e

Pre komunikáciu v ROS-e je ako prvé potrebné spustiť `roscore` inač označovaný aj ako *Master*. Je to centrálny uzol, ktorý si udržiava prehľad o ostatných uzloch a zabezpečuje vytvorenie komunikáciu medzi nimi. Po čo uzol nadviaže komunikáciu s druhým uzlom už nepotrebuje komunikovať ďalej s *Master-om*.

V ROS-e sa používajú dva druhy komunikácie a to:

- **Asynchrónne –** Uzly môžu medzi sebou komunikovať správami, ktoré sú distribuované skrz zdieľaný kanál nazývaný témy. Uzly sa prihlasujú na témy ako *publisher* resp. *subscriber* a tým môžu zasielať resp. prijímať správy z danej témy. Platí že na jednu tému môže byť viacero *publisher-ov* ako aj *subscriber-ov*.
  `roscore` s inými uzlami komunikuje pomocou protokolu vychádzajúceho z XML-RPC.

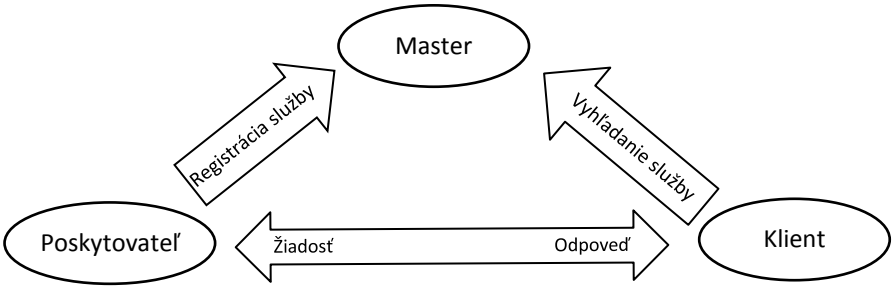


Obrázok 2.2: Schéma asynchrónnej komunikácie medzi uzlami v ROS-e[1]

Ako prvé uzol oznámi *Master* o vytvorení alebo prihlásení sa na tému a pre publikáciu. Podobne sa prihlási u *Master-u* aj druhý uzol pre odber. Následná komunikácia medzi týmito uzlami prebieha bez výpomocí *Master-u*.

[1] https://projets-ima.plil.fr/mediawiki/images/2/23/ROScom.png

- **Synchrónne –** Pre priamu komunikáciu sa používajú služby. Uzol musí najprv oznámiť *Master-u*, že danú službu poskytuje. Ak iný uzol, klient, vyžaduje použitie určitej služby musí od *Maste-u* zistiť adresu danej služby a následne ju priamo zavolať. Po zavolaní služby uzol poskytujúci službu odpovie klientovi a ukonči spojenie. [8]



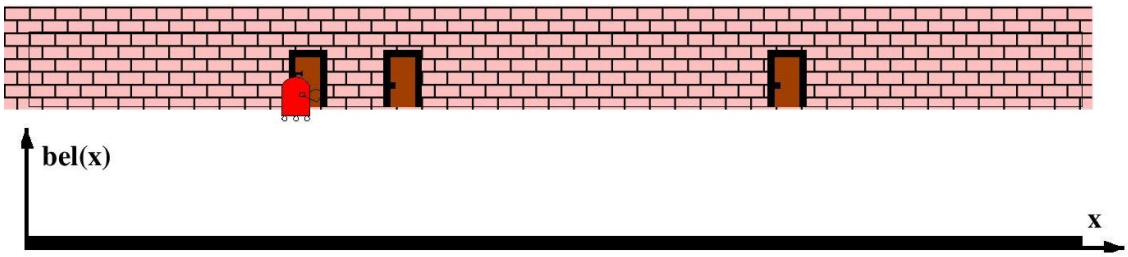
Obrázok 2.3 Schéma používania služby

3 Pravdepodobnostná robotika

Túto problematiku najlepšie popisuje trojica autorov Thrun, Burgarg a Fox v Knihe *Probabilistic Robotics*.[1] Z tejto knihy som čerpal informácie pre túto kapitolu ako aj pre zbytok tejto práce preto ďalej nebudem odkazovať na tento zdroj.

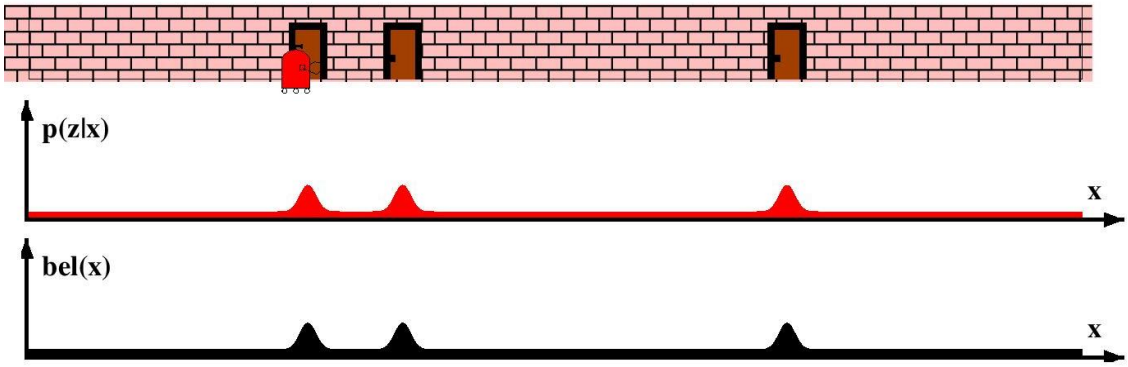
Pravdepodobnostná robotika je spôsob prístupu k robotike, ktorá berie v úvahu neistotu vo vnímaní a činnostiach robota. Kľúčovou myšlienkou pravdepodobnostnej robotiky je vyhodnocovať neistotu výhradne používaním výpočtov pravdepodobnostnej teórie. Thrun píše: „*A robot that carries a notion of its own uncertainty and that acts accordingly, will do better than one that does not.*“[14] Čo v preklade znamená: Robot ktorý dbá na svoju vlastnú neistotu a koná podľa toho, bude mať lepšie výsledky ako ten nie.

Pre lepšie znázornenie, uvediem príklad lokalizácie mobilného robota. Lokalizácia je problém vyhodnocovania súradníc robota v externých referenčných rámcoch zo senzorových dát, na mape prostredia v ktorom sa robot nachádza. Nasledujúce obrázky ilustrujú riešenie tohto problému podľa pravdepodobnostnej robotiky. Tento konkrétny prípad sa nazýva *globálna lokalizácia*, kde robot je vložený do priestoru o ktorom nemá žiadne informácie. V pravdepodobnostnom paradigme sa aktuálne vyhodnotenie nazýva *belief*, je vyobrazené ako funkcia hustoty pravdepodobnosti pre všetky pozície na mape. To nám zobrazuje obrázok 2.1, ktorý zobrazuje rovnomerné rozloženie zodpovedajúce maximálnej neistote.



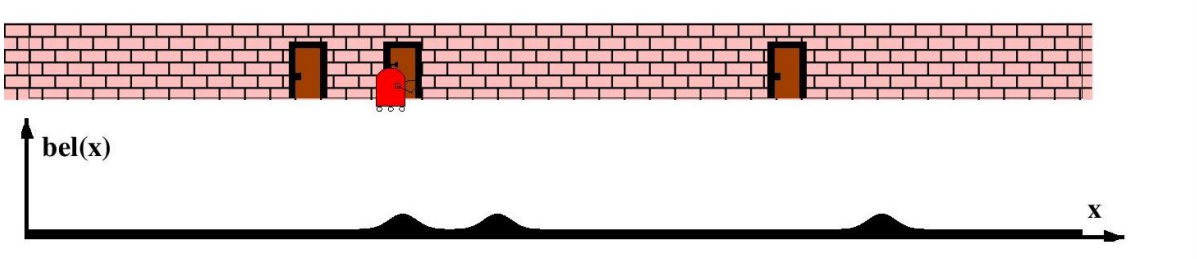
Obrázok 3.1 Počiatočná poloha robota v neznámom prostredí

Povedzme, že robot vykonal meranie a zistil, že sa nachádza vedľa dverí. Zmena sa premietne do belief-u pridaním vysokej pravdepodobnosti na lokácie s dverami a nízkej pravdepodobnosti kdekoľvek inde.



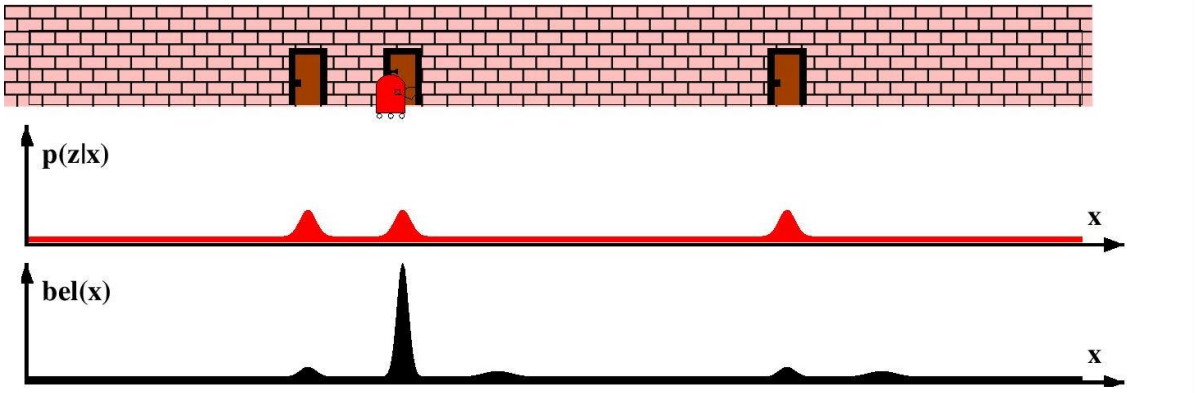
Obrázok 3.2 Zmena v predpokladanej polohe po prvom meraní

Keďže ale v našom prostredí sa nachádzajú troje dvere robot predpokladá, že môže stáť pred ľubovoľnými z nich. Z dôvodu neistoty v senzoroch, aj nedverové lokácie môžu nadobúdať nenulové hodnoty. Obrázok 2.3 nám zobrazuje vplyv pohybu robota na belief. Belief sa posunie v smere pohybu robota. Taktiež sa trochu znížia pravdepodobnosti z dôvodu neistoty pri pohybe.



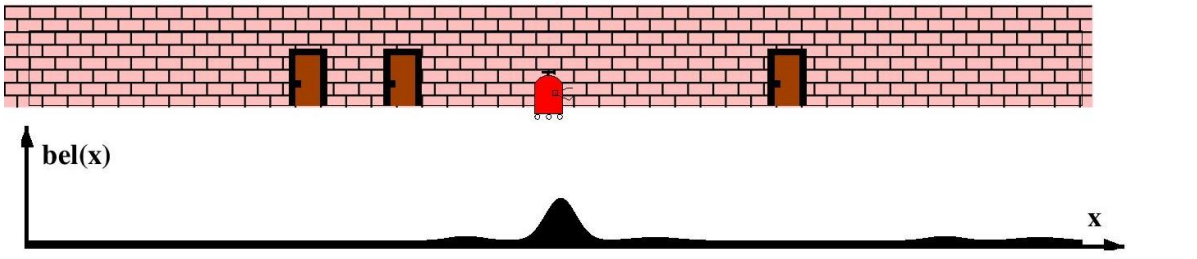
Obrázok 3.3 Pohyb robota

Po následnom nasnímaní zistí blízkosť dverí s následným dopadom na belief ako zobrazuje obrázok 2.4. Algoritmus pridá pravdepodobnosti na lokácie s dverami a robot si už začína byť vcelku istý kde sa nachádza.



Obrázok 3.4 Dopad nameraných hodnôt na belief po pohybe

Pri ďalšom pohybe je robot sa jeho odhad znižuje avšak stále násobne prevyšuje ostatné hodnoty. V príklad sme si predviedli problém vnímania ako problém vyhodnotenia pozície a ako lokalizačný algoritmus sme použili *Bayesov filter*.



Obrázok 3.5 Belief robota po ďalšom pohybe

3.1 Neistota v robotike

Robotika je veda o vnímaní a manipulovaní s fyzickým prostredím skrz počítačom riadeným mechanickým zariadením. Robotické systémy majú spoločnú myšlienku a to že sa nachádzajú vo fyzickom svete, vnímajú prostredie pomocou senzorov a pracujú s prostredím pohyblivými časťami robota.

Neistota vzniká ak robot má nedostatok podstatných informácií pre prevedenie jeho úlohy. Táto neistota vzniká na základe piatich rôznych faktoroch:

1. **Prostredie.** K fyzickému svetu neoddeliteľne patrí jeho nepredvídateľnosť. Aj keď úroveň neistoty v upravenom prostredí ako výrobné pásy je nízka, prostredia ako napríklad diaľnice alebo obytné priestory sa dynamicky menia a sú nepredvídateľné.
2. **Senzory.** Každý senzor je stavaný tak aby vnímal iba určité dáta. Najväčšie dopad na obmedzenia senzorov majú dva faktory. Prvý, vzdialenosť a rozlíšenie senzorov na základe fyzikálnych zákonov. Za druhé, senzory sú vystavené šumom, ktoré rušivo pôsobia na merania, nepredvídateľným spôsobom a tým vyznačujú hranice pre informácie, ktoré dokážeme získať zo senzorových meraní.
3. **Robot.** Ovládanie robotov s motormi je, aspoň do určitej miery, nepredvídateľné. Dôvodom je šum ovládania a opotrebenia. Niektoré ovládania, ako napríklad priemyselné robotické ramená, sú vcelku presné. Naopak nízkorozpočtové pohyblivé roboty môžu byť extrémne nepresné.
4. **Model.** Model je abstrakcia reálneho sveta. Pre zjednodušenie sa modelujú hlavne dôležité aspekty, zatiaľ čo tie menej dôležité sa vypúšťajú. Takýto model sa stáva do určitej miery nepresným. Nedostatky modelu sú zdrojom neistoty, ktorá je v robotike do veľkej miery ignoruje.
5. **Výpočty.** Roboti pracujú v reálnom čase, čo obmedzuje množstvo času použiteľný na výpočty. Veľa najmodernejších algoritmov sú aproximované a získavajú lepšiu časovú odozvu za cenu presnosti.

Všetky tieto faktory zvyšujú neistotu v robotike. Bežne sa neistota v robotike ignoruje ale jak sa postupne roboti dostávajú z im prispôsobeného prostredia do viac a viac nepredvídateľného, schopnosť zvládať neistotu je zásadnou pre tvorbu úspešných robotov.

4 Merania zo senzorov

Robot, ktorý chce mať povedomie o svojej polohe (stave) potrebuje byť vybavený prístrojmi, ktoré snímajú jeho okolie a jeho samotného. Tieto prístroje nazývame senzory. Senzory ktoré snímajú okolie, ako napríklad sonar, lidar alebo kamera, umožňujú robotovi získať informácie o stave okolitého prostredia. Tieto informácie môže využiť k jeho navigácií prostredím tak aby nedošlo ku nechceným kolíziám alebo aj ku lokalizácií jeho samotného. Druhým spôsobom ako robot udržiava informácie o svojom stave sledovania svojho riadenia a meraní zo senzorov napr. rýchlosti. Sledovaním riadenia môže byť získavanie informácií z enkodéru motora kolies robota, ten nám podáva správu o vzdialenosti, ktorú dané koleso prešlo za časový úsek.

Každý senzor, ktorý chceme používať na určovanie polohy robota musíme nastaviť tak aby posielal správy cez ROS. Každá z týchto správ nesie informáciu z jedného senzoru zabalené v štruktúre vhodnej k zasielaniu. Generovanie týchto správ prebieha priamo v počítači robota (napr. Arduino).

K odosielaniu správ cez ROS sú určené publishery. Ako prvé treba oznámiť ROS-u, že budeme posielať správy na určitý topic. ROS vytvorí daný topic a následne môže publisher posielať vygenerované správy na topic. Zasielanie prebieha periodicky v určitom časovom intervale tak aby informácie zo senzorov boli vždy čo najviac aktuálne.

Na prijímanie správ potrebujeme vytvoriť subscriber. Ten sa „podpíše“ na prijímanie správ z určeného topicu. subscriber musí mať určenú funkciu, ktorú bude volať keď dostane správu z topicu. Takéto funkcie sa nazývajú „callback“. Samotný subscriber musí byť v pohotovosti na prijímanie správ. Na to slúži napríklad `ros::spin()`. Táto funkcia v podstate cykluje program a pri zmene na niektorom z topicov, ktoré majú v programe svoj subscriber, volá callback daného subscribera.

4.1 Transformácia robota

Pre správne vyhodnocovanie správ zo senzorov je potrebné vedieť kde sa dané senzory nachádzajú a ako sú nasmerované vzhľadom na robota. Napríklad ak dostaneme informáciu zo senzoru o prekážke, ktorá je pol metra pred senzorom, stále nevieme kde sa daná prekážka nachádza (vpredu, vzadu, napravo, atď.). Ale ak máme informáciu, že daný senzor je na prednej časti robota a smeruje dopredu spresní to informáciu o prekážke a zistíme že prekážka je pol metra **pred** robotom.

Robot sa skladá z veľkého množstva častí, jak pohyblivých tak statických, je potrebné týmto častiam nastaviť ich pozíciu v súradnom systéme. Tieto systémy nazývame rámce. Robot v základe rozlišuje niekoľko rámcov (súradných systémov). Najdôležitejším z nich je rámec sveta často označovaný anglickým slovom „world“. Na tento rámec priamo nadväzuje rámec základne robota. Ďalšie rámce potom môžu byť rámce senzorov, ktoré robot používa. Tieto rámce vytvárajú stromovú štruktúru, kde rámec sveta je najvyššie postaveným rámcom. Podmienka je, že každý rámec môže mať len jediného rodiča. Môžu ale zato mať ľubovoľný počet potomkov.

Vytváranie rámcov je možné pomocou transformačného broadcasteru kde nastavíme posun a orientáciu každého rámca vzhľadom na rámec rodičovský. Tieto informácie následne vysiela do ROS-u na topic transformácie. Naslúchanie potom prebieha podobne ako pri senzoroch s tým, že potrebujeme transformačný listener. Ten nám podľa jeho nastavenia získava z topicu transformácií, informácie o požadovanom rámci.

Ak použijeme túto metódu na všetky merania, dokážeme určiť pozíciu každého bodu v priestore jak oproti robotovi tak aj v súradnom systéme mapy.

4.2 Odometria

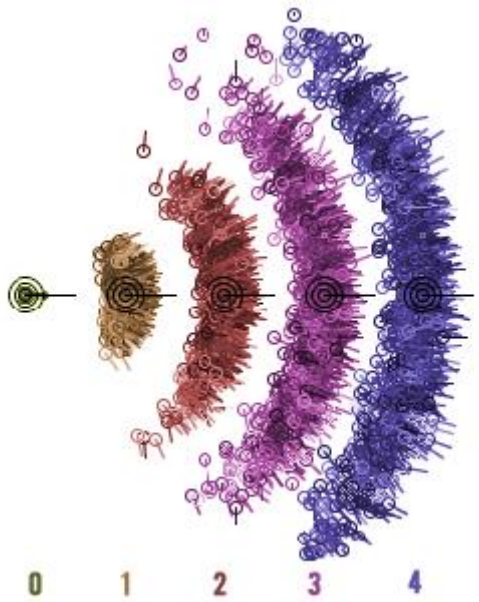
Odometria je relatívna metóda lokalizácie založená na odhade zmeny pozície a orientácie kolesového robota prostredníctvom údajov o otáčaní jeho hnacích alebo bežných kolies nameraných pomocou rotačných enkodérov.

Rotačný enkodér je zariadenie prevádzajúce pohyb na ďalej spracovateľný elektrický signál. Podľa konštrukcie a funkcionality môžeme enkodéry rozdeliť na jednokanálové, inkrementálne a absolútne. [13]

4.2.1 Chyby odometrie

Odometrie a je založená na predpoklade, že nameraný rotačný pohyb kolesa je možne prepočítať na posun robota v rovine. Nedokonalé splnenie tohto predpokladu je zdrojom chýb odometrie. Chyby odometrie môžeme rozdeliť na systematické a nesystematické.

Systematické chyby sú spôsobené drobnými nepresnosťami a ich veľkosť je dobre zhora odhadnuteľná; celková naakumulovaná chyba rastie lineárne so vzdialenosťou, ktorú robot prejde a dá sa s ňou pri odhade pozície robota počítať.



Obrázok 4.1 Ilustrácia systematickej chyby [13]

Naproti tomu nesystematické chyby sa vyskytujú náhodne, nedajú sa dobre predvídať a ich veľkosť sa dá len ťažko odhadnúť. V najhoršom prípade môže ich dôsledkom dôjsť až ku úplnej strate orientácie robota. Kvôli svojej nepredvídateľnosti sa s nimi nedá vôbec počítať avšak niektoré z nich sa dajú aspoň detekovať.[13]

4.2.2 Posielanie odometrických správ

Správy prispôsobené na posielanie odometrických informácií obsahujú hodnoty vyjadrené v základných súradných systémoch a to pozíciu, orientáciu, rýchlosť a uhlovú rýchlosť v x, y, z osiach. Je možne prijať aj hodnoty vzdialenosti a smeru (uhol), v základe to ale znamená iba prepočet týchto hodnôt tak aby sme dostali štandardný odometrický vektor hodnôt.

Samozrejme aj odometriu je potrebné transformovať aby zodpovedala potrebnému rámcu. Prevažne to bude na rámec sveta (mapy v ktorej sa robot pohybuje). [8]

4.3 Fúzia meraní zo senzorov

Najprv uvediem na správnu mieru terminológiu v systémoch fúzie. Termíny „fúzia senzorov“, „fúzia dát“, „fúzia informácií“, „viac-senzorová fúzia dát“ a „viac-senzorová integrácia“ sa používajú v rôznych odvetviach, technikách, technológiách, systémoch a aplikáciách týkajúcich sa získavania dát z viacerých zdrojov informácií. Aby sa zamedzilo nedorozumeniam v terminológií, Dasarathy sa rozhodol používať termín „fúzia informácií“ ako termín zastrešujúci fúziu dát akéhokoľvek druhu (ako cituje Elmenreich [15]). Tento termín ale je pre naše použitie príliš všeobecný, preto použijem termín „fúzia senzorov“ ktorej definícia znie: Fúzia senzorov je kombinácia dát zo senzorov a dát odvodených z senzorových dát. Ako výsledkom je informácia, ktorá je určitým spôsobom presnejšie alebo lepšie ako informácie z každého zdroja samostatne. [15]

Dôvod fúzie senzorov je teda zlepšenie výsledkov zo získaných informácií (resp. meraní). Je niekoľko výhod systémov s fúziou oproti systémom len s jediným senzorom. Obecne, merania z fyzických senzorov trpia istými problémami:

Stratovosť senzoru: Porucha senzoru spôsobuje stratu vnímania objektu.

Obmedzené pokrytie priestoru: Senzory väčšinou pokrývajú len obmedzený priestor.

Obmedzené časové pokrytie: Senzory pracujú na určitej frekvencií.

Nepresnosť: Presnosť senzorov je obmedzená podľa snímaného objektu.

Neistota: Je protikladom nepresnosti. Záleží skôr na sledovanom objekte než na snímacom zariadení. Systém s jediným senzorom si nedokáže znížiť neistotu z dôvodu limitovaného pohľadu na objekt.

Tieto nedostatky sa dajú výrazne zlepšiť použitím viacerých senzorov v systéme a ich fúziou.

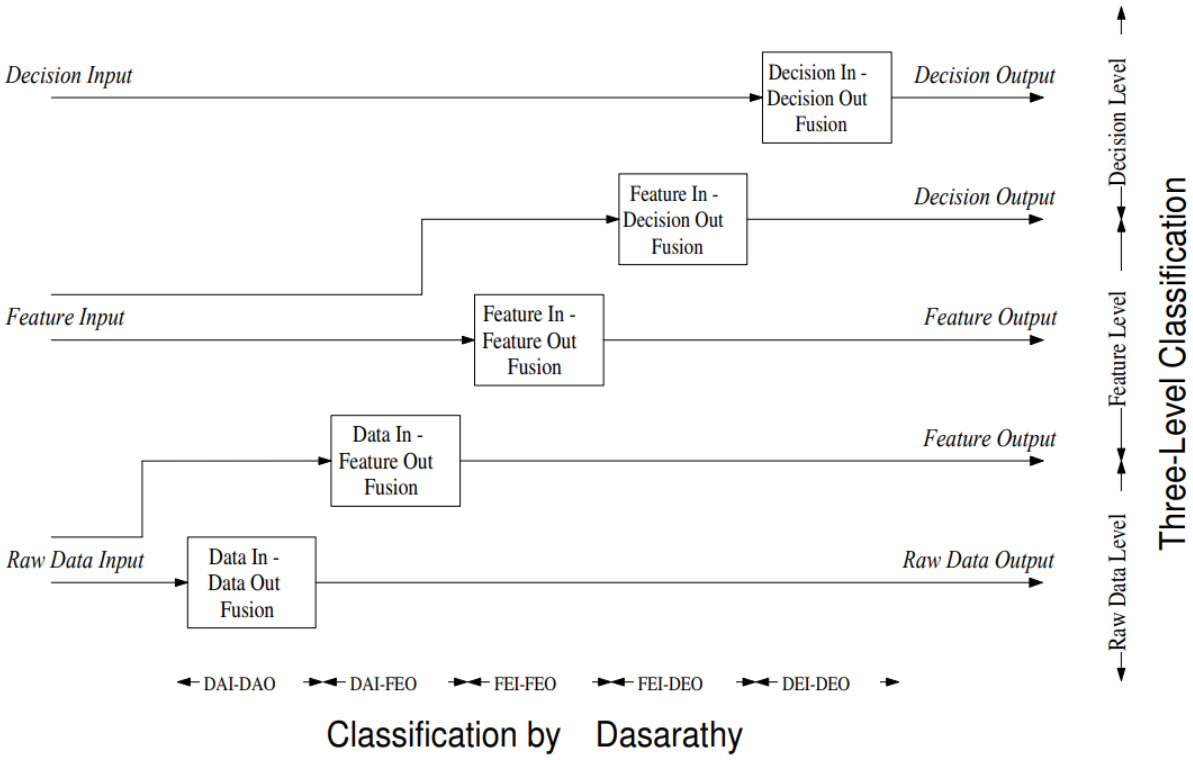
4.3.1 Typy fúzie senzorov

Fúzia senzorov sa dá rozdeľuje podľa niekoľkých kategórií:

Kategorizácia podľa úrovne:

- **Nízkoúrovňová fúzia** – (raw data fusion) Kombinuje nespracované dáta (raw data) z viacerých zdrojov a vytvorí z nich nové dáta, ktoré by mali podávať lepšie informácie ako pôvodné.
- **Fúzia strednej úrovne** – (feature level fusion) Kombinuje rôzne vlastnosti a vytvára z nich mapu, ktorá môže byť použitá na detekciu a segmentáciu.
- **Vysokoúrovňová fúzia** – (decision) Kombinuje rozhodnutia od viacerých expertov. Metódy rozhodovania zahŕňajú štatistické metódy, hlasovanie (voting), fuzzy logiku.

Kategorizácia podľa vstupov/výstupov – je kategorizácia do prechádzajúceho trojúrovňového modelu podľa vstupov a výstupov fúzie. Dôvod Dasarathyho modelu je existencia fúzneho paradigmu kde vstup a výstup patria do rôznych úrovní. [15]



Obrázok 4.2 Alternatívna charakterizácia fúzie podľa vstupu a výstupu [15]

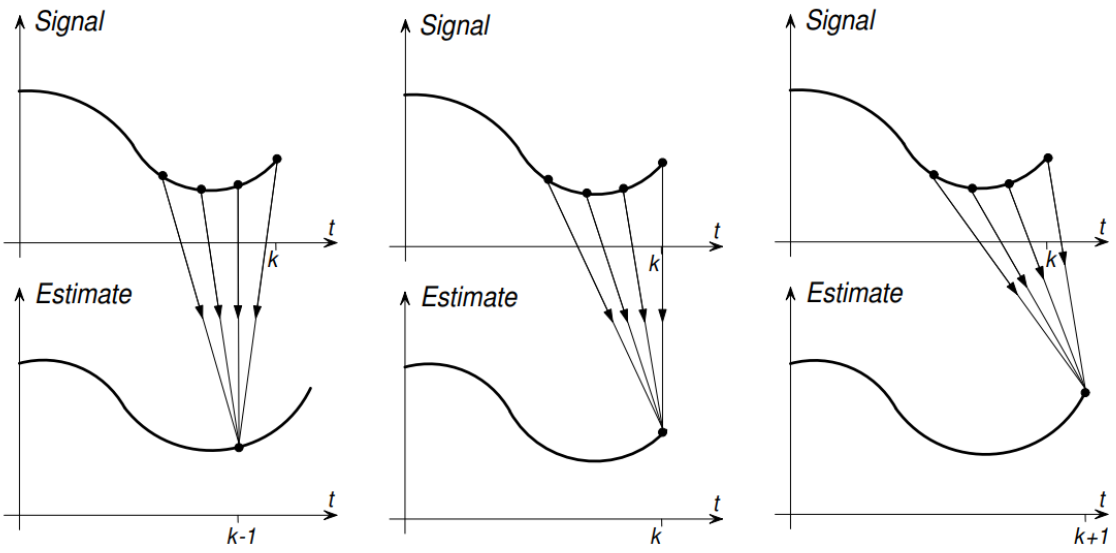
Kategorizácia podľa nastavania senzorov:

- **Komplementárne -** Je konfigurácia senzorov kde senzory nie sú na sebe priamo závislé, ale môžu byť skombinované aby sa dosiahlo ucelenejšieho obrazu objektu/prostredia ktoré senzory sledujú.
- **Konkurenčné –** Ak senzory nezávisle na sebe poskytujú informácie o tom istom objekte.
- **Kooperatívne –** Na sebe nezávislé senzory poskytujú informácie z ktorých je možné odvodiť inú informáciu ktorú by nebolo možné získať za použitia jediného senzoru.

4.3.2 Metódy a aplikácie

Obecne základnou úlohou senzorov je poskytovanie informácií ohľadom prostredia meraniami. Pretože merania obsahujú šum potrebujeme viacero meraní nato aby sme zrekonštruovali hľadaný parameter. Podľa času vyhodnocovaného odhadu môžeme rozlišovať tri prípady:

- **Vyhladzovanie –** pre každý vyhodnocovaný bod potrebujeme merania z niekoľkých predchádzajúcich, aktuálneho a nasledujúcich meraní z ktorých sa vyhodnotí daný bod. Merania musia byť v reálnom čase ale samotné vyhladzovanie môže prebiehať offline.
- **Filtrovanie** – Aktuálny stav je vyhodnocovaný z aktuálneho merania a z predchádzajúcich stavov. Filtrovanie prebieha v reálnom čase.
- **Predpoveď –** Aktuálny stav sa vyhodnocuje zo série predchádzajúcich meraní. Predpoveď potrebuje adekvátny systémovy model aby vyhodnocoval zmysluplné odhady. Predpoveď pracuje v reálnom čase.



(a) (b) (c)

Obrázok 4.3 Vyhladzovanie (a), Filtrovanie (b), Predpoveď (c) [15]

Veľa filtračných algoritmov pokrýva všetky tri aspekty. Filtrovanie a predpoveď je základnou súčasťou každého sledovacieho systému. Využívajú odhad aktuálnych a budúcich kinematických hodnôt ako pozícia, rýchlosť a zrýchlenie.

5 Rekurzívny odhad stavu

Jadrom pravdepodobnostnej robotiky je v odhade stavu robota pomocou senzorových dát. Odhadovanie stavu robota odkazuje na problém s vyhodnocovaním hodnôt zo senzorových dát, ktoré nie sú priamo viditeľné ale dajú sa odvodiť.

5.1 Interakcia robota s prostredím

Prostredie (alebo svet) v ktorom sa robot nachádza je dynamický systém, ktorý vlastní nejaký vnútorný stav. Informácie o tomto stave sa robot dozvedá pomocou senzorových meraní. Ale pretože senzory obsahujú určitú nepresnosť (šum) a často sa niektoré informácie ani priamo nedajú nasnímať, robot si udržiava svoj *belief* s ohľadom na stav prostredia.

5.1.1 Stav

Prostredia sú charakterizované pomocou stavu. Stav predstavuje kolekciu všetkých aspektov robota a prostredia, ktoré môžu ovplyvniť budúcnosť. Stavy môžu byť dynamické, ako pozícia ľudí, alebo statické, ako pozícia stien budovy. Stav obsahuje aj hodnoty týkajúce sa samotného robota, ako jeho polohu, rýchlosť a ďalšie. Ďalej budem stav popisovať písmenom x, kde stav v konkrétnom čase x_t. Teraz priblížim stavy, ktoré budem používať:

- Postoj robota, ktorý obsahuje jeho lokáciu a orientáciu vzhľadom na globálny koordinačný rámec. Stav rigidného pohyblivého robota sa skladá zo šiestich premenných, tri sú pre ich karteziánske koordináty a tri pre uhlovú orientáciu (Eulerové uhly).
- Konfigurácia akčných členov robota, ako kĺby pohyblivých častí robota. Kde každý stupeň voľnosti je charakterizovaný jedno-dimenzionálnou konfiguráciou v ľubovoľnom čase.
- Rýchlosti robota a rýchlosti v jeho kĺboch. Pohyb rigidného robota v priestore je charakterizovaný až šiestimi rýchlostnými premennými. Jedna pre každú premennú postoja robota.
- Poloha a vlastnosti okolitých objektov v prostredí. Tieto objekty môžu byť napríklad stromy, steny, alebo pixel na nejakej ploche. Vlastnosti takýchto objektov môžu byť napríklad ich vzhľad. Podľa granularity modelovaného stavu môže prostredie robota obsahovať od niekoľkých desiatok po milióny stavových premenných.
- Lokácie a rýchlosti pohyblivých objektov.
- Ďalšie špecifické stavové premenné. (stav batérie, poškodenie senzoru, atď.)

5.1.2 Interakcia s prostredím

Interakcia medzi robotom a prostredím má dva základné spôsoby: Robot ovplyvňuje prostredie pomocou jeho pohyblivých častí. A robot zbiera informácie o prostredí pomocou senzorov. Tieto spôsoby môže vykonávať súčasne.

Senzorické merania. Proces vnímania okolia, robot vykonáva senzormi, ktoré mu dodajú informácie o prostredí. Výsledok tohto procesu sa nazýva meranie (angl. measurement). Tieto merania ale prichádzajú s určitým oneskorením.

Riadiace akcie menia stav prostredia. Aktívne aplikujú silu na prostredie okolo robota. Napríklad aj pohyb robota. Aj keď robot samotný nevykonáva žiadnu akciu tak stav sa prevažne zmení. Pre konzistenciu preto predpokladáme že robot nejakú akciu vykonáva vždy.

Podobne ako sú dva typy interakcie s prostredím ma robot prístup ku dvom rôznym dátovým tokom.

Dáta o meraní poskytujú informácie o aktuálnom stave prostredia. Pri väčšine prípadoch sa oneskorenie ignoruje.

Dáta o riadení nesú informáciu o zmene stavu v prostredí. Typickým príkladom je rýchlosť robota. Alternatívne zdroje týchto dát je *odometre*. Odometre sú senzory, ktoré merajú otáčanie kolies robota. Za predpokladu že robot vždy vykonáva riadiacu akciu môžeme povedať, že máme presne jeden záznam o riadiacej akcií za daný krok v čase.

Rozdiel medzi meraním a riadením je dôležitý, pretože každý hrá inú rolu. Meranie poskytuje zisk informácií o prostredí a tým zvyšuje znalosti robota zatiaľ čo riadením skôr stráca znalosti z dôvodu šumu v pohyblivých častiach robota a náhodnosti prostredia. Aj keď niekedy je si robot viac istý svojim stavom pravé z riadiacich akcií. Rozdelenie týchto akcií výhodné pre prácu ale reálne sa vykonávajú súbežne.

5.1.3 Pravdepodobnostné generatívne zákony

Vývoj stavu a meraní je riadený pravdepodobnostnými zákonmi. Vo všeobecnosti, stav v čase x_t je generovaný náhodne. Takže vznik x_t môže byť ovplyvnený všetkými predchádzajúcimi stavmi, meraniami a riadením. Preto pravdepodobnostný zákon môže charakterizovať vývoj stavu ako rozdelenie pravdepodobnosti vo tejto forme:

$$\boldsymbol{p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t})} \tag{5.1.1}$$

Ale ak stav *x* je úplný, je potom dostačujúcim zhrnutím všetkých predchádzajúcich krokov. Obzvlášť x_{t-1} je dostačujúcim údajom o všetkých predchádzajúcich riadeniach a meraniach až do tohto bodu v čase, čiže $u_{1:t-1}$ a $z_{1:t-1}$.

$$\boldsymbol{p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t \mid x_{t-1}, u_t)} \tag{5.1.2}$$

Podobne môžeme modelovať proces generovanie meraní, za predpokladu že x_t je úplné.

$$\boldsymbol{p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(z_t \mid x_t)} \tag{5.1.3}$$

Inač povedané, stav x_t je dostatočný k predpovedi merania z_t, za predpokladu úplnosti x_t.

Rovnica 5.1.1 nazývaná *pravdepodobnosť zmeny stavu* a Rovnica 5.1.2 nazývaná *pravdepodobnosť merania* dohromady popisujú dynamický systém robota a jeho prostredia.

5.1.4 Rozloženie pravdepodobnosti pre *belief*

Belief je ďalším kľúčovým konceptom v pravdepodobnostnej robotike. Predstavuje vnútornú vedomosť robota o stave prostredia. Keďže stav robota sa nedá merať priamo, rozlišujeme preto jeho reálny stav a jeho belief alebo stav podľa vedomosti.

V pravdepodobnostnej robotike sa reprezentuje belief pomocou rozloženia podmienenej pravdepodobnosti. Rozloženie pravdepodobnosti pre belief priradí pravdepodobnosť pre každú možnú hypotézu s ohľadom na reálny stav. Rozloženie belief-u sú posteriórne pravdepodobnosti nad stavovými premennými, ktoré sú podmienené dostupnými dátami.

$$\boldsymbol{bel(x_t) = p(x_t \mid z_{1:t}, u_{1:t})} \tag{5.1.4}$$

Tento posterior je rozloženie pravdepodobnosti nad stavom x_t v čase *t*, podmienené všetkými predchádzajúcimi meraniami $z_{1:t}$ a všetkými predchádzajúcimi riadeniami $u_{1:t}$.

Niekedy sa ukáže ako potrebné vypočítať posterior pred meraním z_t, v takom prípade bude rovnica upravená takto:

$$\boldsymbol{\overline{bel}(x_t) = p(x_t \mid z_{1:t-1}, u_{1:t})} \tag{5.1.5}$$

Takéto rozloženie pravdepodobnosti sa nazýva *predpoveď* v kontexte pravdepodobnostného filtrovania. Výpočet $bel(x_t)$ z $\overline{bel}(x_t)$ je nazývane *korekcia* alebo *aktualizácia merania*.

5.2 Bayesove filtre

Najviac všeobecný algoritmus pre výpočet belief-u udáva algoritmus Bayesovho filtru. Tento algoritmus vypočítava rozloženie belief - u bel z meraní a riadiacich dát.

Tabuľka 5.1 znázorňuje jednoduchý pseudo-algoritmus Bayesovho filtra. Bayesov filter je rekurzívny, čo znamená, že belief v čase *t* je vypočítaný z belief-u v čase *t – 1*. Jeho vstup je belief v čase *t – 1*, súčasne s najnovším riadením u_t a najnovším meraním z_t. Jeho výstup potom je belief z času *t*. Tabuľka popisuje iba jeden krok tohto algoritmu a to *aktualizačné pravidlo.*

Algoritmus Bayesovho filtra obsahuje dva dôležité kroky. Na Riadku 3, spracováva riadenie u_t. Čo robí vypočítaním belief-u podľa stavu x_t na základe predchádzajúceho belief-u podľa stavu x_{t-1} a riadenia u_t. Konkrétne belief $\overline{bel}(x_t)$, ktorý robot priradí stavu x_t je získaný integrovaním súčinu dvoch rozložení: to čo bolo priradené do x_{t-1} predtým, a pravdepodobnosť ktorou riadenie u_t indukuje prechod z x_{t-1} do x_t. Tento aktualizačný krok sa nazýva *predpoveď*.

Druhý krok Bayesovho filtra sa nazýva aktualizácia meraním. Na Riadku 4 algoritmu Bayesovho filtra násobí belief $\overline{bel}(x_t)$ pravdepodobnosťou že meranie z_t mohlo byť zaznamenané. A to robí pre všetky posteriorne stavy x_t. Výsledok sa normalizuje pomocou normalizačnej konštanty η . A to nás dovedie k finálnemu belief-u $bel(x_t)$, ktorý na Riadku 6 vrátime.

Aby mohol byť vypočítaný posterior belief rekurzívne, potrebuje algoritmus mať počiatočný belief $bel(x_0)$ v čase $t = 0$. V praxi sa bežne používajú prípady plnej vedomosti, kde presne poznáme počiatočny stav, alebo úplnej ignorancie, kde počiatočný stav je uniformným rozložením na x_0.

```
1.    Algoritmh Bayes_filter(bel(x_{t-1}), u_t, z_t):
2.        for all x_t do
3.            bel̄(x_t) = ∫ p(x_t | u_t, x_{t-1}) bel(x_{t-1}) dx
4.            bel(x_t) = η p(z_t | x_t) bel̄(x_t)
5.        endfor
6.        Return bel(x_t)
```

Tabuľka 5.1 Všeobecný algoritmus pre Bayesove filtre

Bayesov filter je ale v tejto forme prakticky nepoužiteľný, resp. len na velice jednoduché prípady alebo s veľkým obmedzením stavového priestoru. [14]

Markov predpoklad. Bayesov filter robi *Markov predpoklad*, ktorý hovorí že stav ktorý je úplný je súhrnom všetkých predchádzajúcich stavov. Tento predpoklad implikuje že takýto belief dostatočne reprezentuje minulosť robota.

5.2.1 Reprezentácia a výpočet

V pravdepodobnostnej robotike sú Bayesove filtre implementované niekoľkými rôznymi spôsobmi. Exaktné techniky výpočtov belief-u sa dajú použiť len vo velice špecializovaných prípadoch. V robotike pri všeobecných problémoch sa musí belief aproximovať.

Pri výbere aproximácie je nutné sa rozhodnúť aké vlastnosti potrebujeme:

1. **Efektivita výpočtov.** Niektoré aproximácie, ako Gaussová aproximácia, umožňujú výpočet belief-u s polynomiálnov časovou zložitosťou. Iné môžu mať exponenciálnu zložitosť. Časticové techniky majú *any-time* (ľubovoľnú) časovú zložitosť, umožňujúci im robiť kompromis medzi presnosťou a efektivitou výpočtov.
2. **Presnosť aproximácie.** Niektoré aproximácia, ako Gaussová aproximácia, sú limitované na jednovrcholové (angl. unimodal) rozdelenia, kde napríklad histogramové reprezentácie môžu aproximovať viacvrcholové (angl multi-modal) rozdelenia ale zo zníženou presnosťou. Časticové reprezentácie dokážu aproximovať širokú škálu rozdelení ale pre veľký poče častíc na to potrebných môže byť obrovská nepresnosť.
3. **Jednoduchosť implementácie.** Náročnosť implementácie ovplyvňuje viacero faktorov ako forma pravdepodobnosti pri meraní alebo pravdepodobnosť prechodu stavu.

5.3 Gaussove filtre

Gaussove filtre je označenie skupiny filtrov pre rekurzívne odhadovanie stavu. Predstavujú najskoršiu implementáciu Bayesovho filtra v spojitom systéme.

Základnou myšlienkou všetkých Gaussových techník je reprezentácia belief-u viacrozmerným normálnym rozdelením . Gaussiany môžu byť reprezentované dvomi spôsobmi:

- **Momentovou reprezentáciou,** ktorá sa skladá zo strednej hodnoty (prvý moment) a kovariancie (druhý moment) Gaussianu.
- **Kánonickou reprezentáciou,** ktorá pozostáva z informačnej matice a informačného vektoru.

Obe tieto reprezentácie sú jedna druhej duálne a môžu byť získané cez inverziu matice druhej reprezentácie.

Bayesov filter môže byť implementovaný pre obidva prípady. Pri momentovej reprezentácií vznikne takzvaný Kalmanov filter a pri kánonickej reprezentácií zase filter informačný. Aktualizácia Kalmanovho filtra na základe riadenia je jednoduché na výpočet, avšak začlenenie merania je komplikovanejšie. Naopak pre informačný filter je začlenenie merania jednoduché a aktualizácia riadenia je náročná.

Pre korektný výpočet posterioru musia obidva filtre splniť tri predpoklady: Počiatočný belief musí byť Gaussian. Pravdepodobnosť prechodu stavu musí obsahovať funkciu, ktorá má lineárny argument s pridaným nezávislým Gaussovým šumom. Pravdepodobnosť pri meraní musí tak isto byť lineárna v argumente s pridaným Gaussovým šumom. Systém spĺňajúci tieto predpoklady sa nazýva lineárny Gaussov systém.

5.3.1 Rozšírenie na nelineárne systémy

Predpoklady o lineárnosti prechodu stavu a lineárnosti pri meraní s pridaným Gaussovým šumom sú v praxi splnené len málokedy. Rozšírenie filtru prekonáva predpoklad linearity. Výpočtom tangentu, tangent je lineárny, pre nelineárnu funkciu umožní aplikáciu filtra na danú funkciu. Táto technika sa nazýva Taylorovo rozšírenie. Pre vykonanie Taylorovej expanzie je potrebné vypočítať prvú deriváciu funkcie a jej ohodnotenie v špecifickom bode. Výsledkom tejto operácie je matica tiež známa ako Jacobián. Výsledné filtre sa nazývajú „rozšírené".

Presnosť rozšírení Taylorovej rady záleží na dvoch faktoroch: Stupeň nelinearity v systéme a šírka posterioru (šírka Gaussovho belief-u). Rozšírené filtre dokážu priniesť dobré výsledky ak je stav systému známy a to s relatívne veľkou presnosťou, takže zvyšná kovariancia je malá.

Najhlavnejšou výhodou Gaussových filtrov je ich polynomiálna časová zložitosť.

Pre túto prácu som si zvolil momentovú reprezentáciu a teda Kalmanov filter a konkrétne jeho rozšírenú verziu. Najprv popíšem jeho lineárnu verziu pre lepšie pochopenie fungovania filtra a potom jeho rozšírenú verziu.

5.4 Kalmanov filter

Kalmanov filter má vysoké využitie v technike. Najbežnejšie použitie je v navigácií, navádzaní a kontrolovaní vozidla a čiastočne v lietadlách. Ďalej je koncept Kalmanovho filtra dosť rozšírený pri analýze údajov závislých od času ako spracovanie signálov alebo ekonometrií. Pre túto prácu najdôležitejším použitím Kalmanovho filtra je plánovanie a kontrola pohybov robota.

Kalmanov filter reprezentuje belief jeho strednou hodnotou a kovarianciou v určitom čase. Ako vstup do Kalmonovho filtra posielame belief a kovarianciu z predchádzajúceho priechodu (resp. z času $t - 1$). Pre aktualizáciu týchto parametrov potrebuje Kalmanov filter hodnoty riadenia a meraní. Výsledkom potom je belief v danom čase reprezentovaný jeho strednou hodnotou a kovarianciou.

Algoritmus pracuje v dvoch krokoch, krok predpovede, kde Kalmanov filter vytvorí odhady premenných v aktuálnom stave spolu s ich neurčitosťou a krok korekcie, kde po vyhodnotení nasledujúceho merania, aktualizuje odhady použitím váženého priemeru (väčšia váha pre odhady s väčšou istotou pravdivosti odhadu). Algoritmus je rekurzívny. Môže fungovať v reálnom čase a pre výpočet potrebuje len aktuálne hodnoty, hodnoty vypočítané v minulom kroku a matice neurčitosti.

Je dôležité uvážiť relatívnu určitosť merania a aktuálneho odhadu stavu, bežne používaným termínom je Kalmanov zisk (gain). Kalmanov zisk je relatívna váha pre meranie a aktuálny odhad stavu a môže byť naladený pre dosiahnutie určitého výkonu. S vysokým nárastom, filter prikladá väčšiu váhu na najnovšie meranie a tým pádom je citlivejší na zmeny. Naopak nízky nárast lepšie nasleduje predpovede modelu.

Pri prevádzaní reálnych výpočtov sú hodnoty odhadu stavu a rozptyl zapísané v matici tak aby sa zvládalo vypočítať v jednej sade výpočtov viacero rôznych dimenzií. Umožňuje to reprezentáciu lineárnych vzťahov medzi premennými v ľubovoľnom prechodovom modeli alebo rozptyle. [3]

5.4.1 Základný model dynamického systému

Kalmanov filter je založený na lineárnom dynamickom systéme s diskrétnym časom. Je modelovaný na Markovom reťazci postavenom na lineárnych operátoroch prerušovaných chybami medzi ktoré patrí napríklad Gaussov šum. Stav systému je reprezentovaný ako vektor reálnych čísel. Každým posunom v diskrétnom čase sa aplikuje lineárny operátor na stav pre vygenerovanie nového stavu s primiešaným

šumom, navyše môže obsahovať informácie z ovládačov systému ak sú známe. Následne ďalší lineárny operátor je zmiešaný s väčším šumom a vytvorí výstup z pravého (skrytého) stavu. Kalmanov filter môže byť považovaný za analogický k skrytému Markovmu modelu. Hlavný rozdiel je že premenné zo skrytého stavu získavajú hodnoty v spojitom priestore. Medzi skrytým Markovým modelom a Kalmanovým filtrom je vysoká dualita. [5]

Nato aby sme použili Kalmanov filter pre odhadnutie vnútorného stavu procesu ak poznáme iba sekvenciu sledovaní so šumom, musíme namodelovať proces zhodný s štruktúrou Kalmanovho filtra. To znamená špecifikovanie nasledujúcich matíc: $\mathbf{F}_k$, model prechodov medzi stavmi; $\mathbf{H}_k$, model pozorovania; $\mathbf{Q}_k$, kovariancia procesového šumu; $\mathbf{R}_k$, kovariancia šumu pri pozorovaní; a občas $\mathbf{B}_k$, model vstupu riadenia, pre každý krok v čase *k*.

Kalmanov filter predpokladá, že pravý stav v čase *k* je odvodený od stavu v čase $(k-1)$.

$$\boldsymbol{x_k = F_k x_{k-1} + B_k u_k + w_k} \tag{5.4.1}$$

Kde

- $\mathbf{F}_k$ je model prechodov medzí stavmi, ktorý je násobený s predchádzajúcim stavom $\mathbf{x}_{k-1}$
- $\mathbf{B}_k$ je model ovládania vstupu, ktorý je násobený vektorom ovládania $\mathbf{u}_k$
- $\mathbf{W}_k$ je procesový šum, ktorý predpokladáme, že je získaný z viacrozmerného normálneho rozloženia vrátane kovariancie $\mathbf{Q}_k$

V čase *k* pozorovanie $\mathbf{z}_k$ pravého stavu $\mathbf{x}_k$ vypočítame ako

$$\boldsymbol{z_k = H_h x_k + v_k} \tag{5.4.2}$$

Kde $\mathbf{H}_k$ je model pozorovania, ktorý mapuje pravý stav priestoru do pozorovaného priestoru a $\mathbf{v}_k$ je šum pozorovania o ktorom predpokladáme, že má nulovú strednú hodnotu Gaussovho bieleho šumu s kovarianciou $\mathbf{R}_k$. [2]

Počiatočný stav a šumové vektory v každom kroku $\{\mathbf{x}_0, \mathbf{w}_1, \ldots, \mathbf{w}_k, \mathbf{v}_1, \ldots, \mathbf{v}_k\}$ sú všetky navzájom nezávislé.

Veľa reálnych dynamických systémov nesedí na tento model presne. Nemodelové dynamiky môžu seriózne znížiť výkonnosť filtra, aj keď údajne dokáže pracovať s neznámymi stochastickými signálmi. Dôvod tohto správania, je závislosť nemodelových dynamík na vstup, a tak môže priviesť vyhodnocovací algoritmus k nestabilite (algoritmus diverguje). Na druhú stranu nezávislý signál bieleho šumu nespôsobí divergenciu algoritmu. Rozlišovanie medzi šumom merania a nemodelovými dynamikami je náročné. Spracováva sa v teórií riadenia pod štruktúrou robustného riadenia. [4]

5.4.2 Detail

Kalmanov filter vyhodnocuje odhady rekurzívne. To znamená, že iba odhad stavu z predchádzajúceho časového kroku a aktuálne meranie je potrebné na výpočet odhadu pre aktuálny stav. Oproti technikám, ktoré počítajú odhad po skupinách, nevyžaduje žiadnu históriu pozorovaní ani odhadov. To čo nasleduje je notácia $\hat{x}_{n|m}$, ktorá reprezentuje odhad **x** v čase *n* vzhľadom na pozorovania po čas *m*. $(m \leq n)$. [2]

Stav filtru je reprezentovaný dvoma premennými:

- $\hat{x}_{k|k}$, koncový (posteriori) stav odhadu v čase *k* vzhľadom na pozorovania po čas *k*,
- $P_{k|k}$, matica koncových kovariancií chýb (miera presnosti odhadu stavu)

Kalmanov filter môže byť zapísaný ako jediná rovnica. Väčšinou ale býva rozčlenená do dvoch fáz: „Predpoveď“ a „Aktualizácia“. Fáza predpovede využíva odhad stavu z predchádzajúceho kroku na vyhodnotenie stavu v aktuálnom stave. Tento predpovedaný odhad stavu je známy ako *priori*,

pretože keď vyhodnocuje stav aktuálneho kroku, nezohľadňuje pozorovanie z aktuálneho kroku. V aktualizačnej fáze je aktuálna *priori* hodnota skombinovaná s aktuálnymi informáciami o pozorovaní a vytvoria odhad stavu. Tento vylepšený odhad sa nazýva *posteriori* odhad stavu. [7]

Typicky sa striedajú dve fáze. Predpoveď napreduje stav dokým nenastane ďalšie naplánované pozorovanie a aktualizácia zahŕňajúce pozorovanie. Ak je z nejakého dôvodu pozorovanie neprístupne tak môže byť preskočených niekoľko fáz aktualizácie zatiaľ čo sa prevedie viacnásobná predpoveď, naopak ak je dostupných viacero pozorovaní súčasne, môže sa previesť viacnásobná aktualizácia.

5.4.3 Rovnice a vysvetlenie

Kalmanov filter pracuje v dvoch základných krokoch: predpoveď a aktualizácia. V kroku predpovede pracuje filter s hodnotami z minulého vyhodnotenia a vypočíta predpokladanú pozíciu a kovarianciu. Hodnoty z tohto kroku sa nazýva „priori". Druhý krok, aktualizácia, vezme hodnoty priori a upraví ich podľa nameraných hodnôt. Veľkosť akou namerané hodnoty ovplyvnia výsledok udáva Kalmanov zisk. Optimálny zisk sa vypočíta pomocou kovariancií. V základe určuje význam alebo dôveryhodnosť nameraných hodnôt. Po kroku aktualizácie dostaneme „posteriori" hodnoty, ktoré by mali zodpovedať polohe robota v priestore. [2] [3]

Nepresnosti, ktoré sa môžu počas procesu vyskytnúť nazývame šum (angl. noise). Rozlišujeme dva druhy šumu a to: Procesový šum a šum pri meraní. Šum sa pripočítava do kovariancie. Procesový šum aj pri velice nízkych nenulových hodnotách zlepšiť vyhodnocovanie a napomáha udržať výsledky v norme. Šum pri meraní sa naopak doporučuje nastaviť na vysoké hodnoty aby sa mohol filter inicializovať na správne hodnoty. [3]

Nasledujúce rovnice popisujú fungovanie lineárneho Kalmanovho filtra v oboch krokoch. V prvom kroku, nazývaný predpoveď, rovnica 5.4.3 vyhodnocuje odhad stavu $\hat{x}_k$ na základe predchádzajúceho stavu $\hat{x}_{k-1}$ a aktuálneho riadenia $\boldsymbol{u}_k$, a a b sú konštanty. Druhá rovnica predpovede 5.4.3 popisuje chybu predpovede, ktorá sa vypočítava rekurzívne z predchádzajúcej hodnoty.

V kroku aktualizácia v rovnici 5.4.5 sa vypočíta Kalmanov zisk g_k, ktorý podľa chyby predpovede určí výpovednú hodnotu novo prijatých meraní. Hodnota r predstavuje priemernú nepresnosť senzorov. V rovnici 5.4.6 upravíme odhad z predpovede aktuálnym meraním, ktorého dopad upravíme podľa Kalmanovho zisku. Nakoniec v rovnici 5.4.7 upravíme chybu predpovede pre ďalší krok, ziskom.

Predpoveď:

$$\hat{\boldsymbol{x}}_{\boldsymbol{k}} = \boldsymbol{a}\hat{\boldsymbol{x}}_{\boldsymbol{k-1}} + \boldsymbol{b}\boldsymbol{u}_{\boldsymbol{k}} \quad \textbf{(5.4.3)}$$

$$\boldsymbol{p}_{\boldsymbol{k}} = \boldsymbol{a}\boldsymbol{p}_{\boldsymbol{k-1}}\boldsymbol{a} \quad \textbf{(5.4.4)}$$

Aktualizácia:

$$\boldsymbol{g}_{\boldsymbol{k}} = \boldsymbol{p}_{\boldsymbol{k}}\boldsymbol{c}/(\boldsymbol{c}\boldsymbol{p}_{\boldsymbol{k}}\boldsymbol{c} + \boldsymbol{r}) \quad \textbf{(5.4.5)}$$

$$\hat{\boldsymbol{x}}_{\boldsymbol{k}} = \hat{\boldsymbol{x}}_{\boldsymbol{k}} + \boldsymbol{g}_{\boldsymbol{k}}(\boldsymbol{z}_{\boldsymbol{k}} - \boldsymbol{c}\hat{\boldsymbol{x}}_{\boldsymbol{k}}) \quad \textbf{(5.4.6)}$$

$$\boldsymbol{p}_{\boldsymbol{k}} = (\boldsymbol{1} - \boldsymbol{g}_{\boldsymbol{k}}\boldsymbol{c})\boldsymbol{p}_{\boldsymbol{k}} \quad \textbf{(5.4.7)}$$

Keďže my potrebujeme aby filter spracovával viacero senzorov súčasne a väčšina senzorov odosiela viac ako jednu hodnotu, potrebujeme filter upraviť tak aby bol schopný spracovať viacero hodnôt. Aby sme mohol filter spracovávať viacero hodnôt, zameníme skalárne premenné maticami alebo vektormi. Zmením *a, b, c, p, r* a *g* maticami *A, B, C, P, R* a *G*. Ďalej z premenných (stav, meranie, šum, riadenie) sa stanú vektory. Niekoľko úprav z dôvodu prechodu na matice a výsledné rovnice vyzerajú nasledovne [3] [4]:

Model:

$$\boldsymbol{x_k = Ax_{k-1} + Bu_k} \quad \textbf{(5.4.8)}$$

$$\boldsymbol{z_k = Cx_k + v_k} \quad \textbf{(5.4.9)}$$

Predpoveď:

$$\boldsymbol{\hat{x}_k = A\hat{x}_{k-1} + Bu_k} \quad \textbf{(5.4.10)}$$

$$\boldsymbol{P_k = AP_{k-1}A^T} \quad \textbf{(5.4.11)}$$

Aktualizácia:

$$\boldsymbol{G_k = P_kC^T/(CP_kC^T + R)^{-1}} \quad \textbf{(5.4.12)}$$

$$\boldsymbol{\hat{x}_k = \hat{x}_k + G_k(z_k - C\hat{x}_k)} \quad \textbf{(5.4.13)}$$

$$\boldsymbol{p_k = (I - G_kC)P_k} \quad \textbf{(5.4.14)}$$

5.4.4 Rozšírenie Kalmanovho filtra

Keďže prostredie okolo robota je nelineárne, ako aj jeho riadenie, potrebujem rozšíriť lineárny Kalmanov filter tak aby dokázal túto skutočnosť spracovávať. Preto znova upravím rovnice, výmenou matíc *A* a *B* za funkciu *f* a maticu *C* za funkciu *h*. Avšak s týmito funkciami ich belief už nie je Gaussian. Pre vyriešenie sa musia funkcie linearizovať.

Linearizáciou dostaneme približnú lineárnu funkciu, ktorá v jej strednej hodnote Gaussianu je tanget funkcie *f*. Premietaním Gaussianu skrz lineárnu aproximáciu vytvorí posterior, ktorý je tiež Gaussian. To isté sa prevedie s funkciou *h*. [2]

Technika ktorú rozšírený Kalmanov filter vyžíva na linearizáciu funkcií je Taylorov rozvoj. Taylorov rozvoj vytvorí lineárnu aproximáciu pre funkciu *f* z hodnoty a náklonu *f*. Jej sklon dostaneme pomocou parciálnych derivácií. Maticu s parciálnymi deriváciami nazývame Jacobian. [3]

Teraz si vypíšeme rovnice rozšíreného Kalmanovho filtra a popíšeme si jednotlivé členy rovníc:

Model:

$$\boldsymbol{x_k = f(x_{k-1}, u_k) + w_k} \quad \textbf{(5.4.15)}$$

$$\boldsymbol{z_k = h(x_k) + v_k} \quad \textbf{(5.4.16)}$$

Rovnica **Chyba! Nenalezen zdroj odkazů.** 5.4.15 – Model prechodu stavu, vypočítava nové stavy priori z Jacobianu funkcie *f* a nakoniec sa pripočíta šum. Často sa výsledok funkcie $f(x) = x$ tým pádom aj jej Jacobian je identitná matica. [1] [3]

Rovnica 5.4.16 – Model merania, je vlastne spôsob ktorým prinášame hodnoty z merania do filtra. Pre zjednodušenie sa môžeme na maticu *H* pozerať ako na nejakú „výberovú“ maticu, ktorá určí ktoré stavy budeme vyhodnocovať. [2] [3]

Predpoveď:

$$\boldsymbol{\hat{x}_k = f(\hat{x}_{k-1}, u_k)} \quad \textbf{(5.4.17)}$$

$$\boldsymbol{P_k = F_{k-1}P_{k-1}F_{k-1}^T + Q_{k-1}} \quad \textbf{(5.4.18)}$$

Predpoveď stavu v rovnici 5.4.17 sa vypočíta z Jacobianu funkcie *f*. Rovnica 5.4.18 je výpočet chyby predpovede kde F je Jacobian funkcie f a Q je kovariancia procesového šumu. Čím väčšie hodnoty v Q tým presnejšie bude nasledovať zmeny v dátach. [2]

Aktualizácia:

$$\boldsymbol{G_k = P_k H_k^T \left(H_k P_k H_k^T + R\right)^{-1}} \quad \textbf{(5.4.19)}$$

$$\boldsymbol{\hat{x}_k = \hat{x}_k + G_k\left(z_k - h(\hat{x}_k)\right)} \quad \textbf{(5.4.20)}$$

$$\boldsymbol{P_k = (I - G_k H_k) P_k} \quad \textbf{(5.4.21)}$$

V kroku aktualizácie si potrebujeme vypočítať optimálny Kalmanov zisk G_k (Rovnica 5.4.19). Pre jeho výpočet pripočítavame maticu nepresnosti merania. Čím väčšie hodnoty G_k tým menej bude filter brať v úvahu meranie. Nakoniec upravíme stavový vektor Kalmanovým ziskom ako aj chybu predpovede. Matica I v rovnici je identitná matica. [2] [3]

6 Navigačný zásobník

Túto kapitolu som napísal s pomocou dokumentácie ROS-u a rovnako nazvaným článkom v dokumentácií. [11]

Navigačný zásobník **(anlg. Nvigation stack)** je na konceptuálnej úrovni pomerne jednoduchý. Prijíma vysielané informácie z odometrie a senzorov a ako výstup odosiela ovládanie rýchlosti na základňu pohybu robota. Avšak použitie navigačného zásobníka na svojvoľného robota je komplikovanejšie. Potrebujeme robota, ktorý funguje na ROS-e, potrebujeme aby mal vytvorený správny transformačný strom a senzory musia publikovať správy s meraniami v správnom type ROS správy. Pre zaručenie čo najlepšej funkčnosti navigačného zásobníka je potrebné nakonfigurovať tvar a rozmery robota.

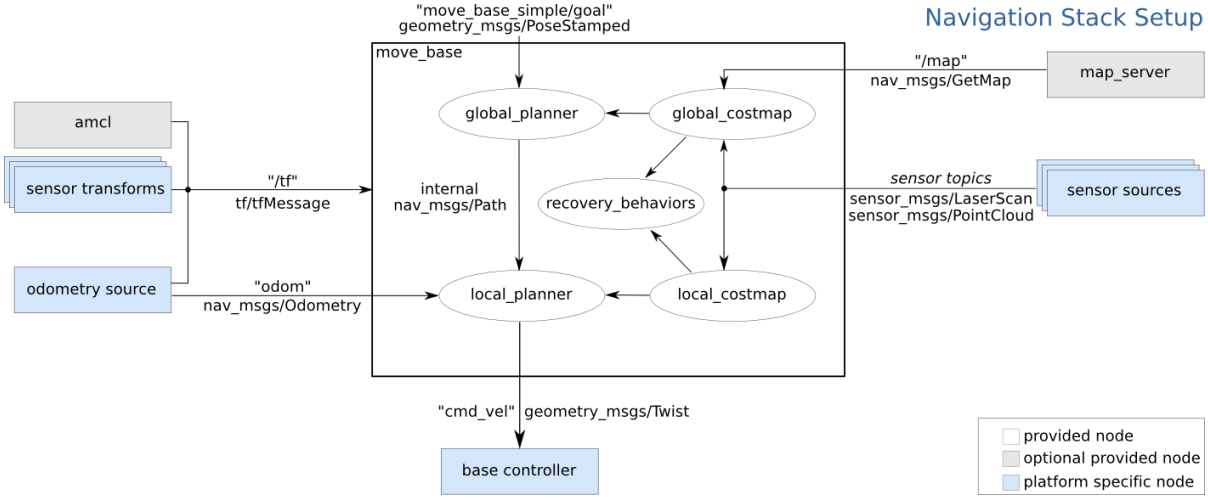
6.1 Hardwarové požiadavky

Aj keď je navigačný zásobník vytvorený tak bol čo naviac všestranný ako len ide, existuje tri hlavné požiadavky na hardware:

1. Je použiteľný jak pre diferenciálny pohon tak aj na holonimický pohon. Predpokladá že základňa pohybu je ovládaná posielaním rýchlostnými príkazmi vo forme: rýchlosť v ose x, rýchlosť v ose y a uhlová rýchlosť theta.
2. Vyžaduje namontovaný rovinný laser niekde na základni pohybu. Využíva sa na mapovanie budov a lokalizáciu.
3. Navigačný zásobník bol vyvinutý na robotovi v tvare štvorca, preto jeho výkon je najlepší na robotoch ktorý majú čo najviac štvorcový alebo okrúhly tvar.

6.2 Nastavenie robota

Navigačný zásobník predpokladá že robot bude nakonfigurovaný konkrétnym spôsobom k tomu aby mohol fungovať. Obrázok 6.1 ukazuje prehľad tejto konfigurácie. Biele komponenty na obrázku sú povinne a sú už implementované, šedé komponenty nie sú povinne ale sú implementované a modré musia byť vytvorené špecificky pre každého robota.



Obrázok 6.1 Nastavenie navigačného zásobníka

6.2.1 Konfigurácia transformácie

Pre navigáciu potrebujeme transformačný strom publikovaný pomocou knižnice `tf`. Transformačný strom popisuje posun a orientáciu jednotlivých častí robota v koordinačných rámcoch a medzi nimi.

Pre vytvorenie transformačného stromu si potrebujeme definovať (namerať) posuny medzi rámcami. Potom vytvoríme vzťah medzi rámcami a uložíme ich do transformačného stromu. V zásade každý uzol v transformačnom strome zodpovedá jednému koordinačnému rámcu a hrany zodpovedajú transformácii, ktorú treba previesť pre presun z aktuálneho uzla do jeho potomka. Stromová štruktúra sa využíva aby medzi ľubovoľnými dvoma koordinačnými rámcami vždy viedla len jedna hrana a predpokladá, že tieto hrany sú vždy orientované smerom od rodiča k potomkovi.

Keď už máme vytvorený transformačný strom potrebujeme tieto informácie vysielať do systému kde budú k dispozícií k naslúchaniu a ich následnému využitiu. Informácie o transformáciách sa posiela na tému (topic) nazývanú `tf`.

Ako prvý musíme vytvoriť uzol, ktorý bude vysielať transformáciu do nášho systému. K tomu si potrebujeme vytvoriť objekt typu `TransformBroadcaster`, ktorý pre odoslanie potrebuje nasledujúcich päť argumentov. Prvý je rotácia rodičovského rámca od potomka. Druhý je posun medzi rámcami. Ako tretie je časová značka. Štvrtý a piaty sú názvy uzla rodičovského a potomka v tomto poradí.

Ďalej potrebujeme uzol, ktorý bude naslúchať dátam vysielaným cez ROS a používať ich k transformácií bodov. Vytvoríme funkciu, ktorá bude transformovať bod, ktorý je vysielaný z rámca potomka a pomocou informácií `tf` vysielaných do systému prevedieme bod do rámca rodiča.

6.2.2 Informácie zo senzorov

Navigačný zásobník využíva informácie zo senzorov tak aby sa robot vyhol prekážkam, ktoré sa nachádzajú v jeho okolí. Správne publikovanie informácií zo senzorov je dôležité pre navigačný zásobník aby mohol bezpečne pracovať.

Predpokladá že senzor budú publikovať správu typu `LaserScan` alebo `PointCloud`. Hlavička takýchto správ sa skladá z troch polí a to z poľa `seq`, ktorý sa po každom poslaní správy automaticky inkrementuje, `stamp` s časovou pečiatkou a `frame-id` pre názov rámcu, ktorý správu vygeneroval. Obsah správy je naplnený dátami z daného senzora.

6.2.3 Odometrické informácie

Navigačný zásobník využíva `tf` na vyhodnotenie pozície robota. Ale z dôvodu, že `tf` neposkytuje informácie o rýchlosti robota, navigačný zásobník potrebuje publikované jak transformácie tak aj odometrické správy cez ROS, ktoré ich obsahujú.

Odometrické správy obsahujú odhad pozície a rýchlosti robota vo voľnom priestore. Hodnota `pose` obsahuje odhad pozície robota v odometrickom rámci. Do hodnoty môžeme pridať neurčitosť pre odhad pozície. Hodnota `twist` potom odpovedá rýchlosti robota v potomkovom rámci. Takisto môže obsahovať neurčitosť odhadu rýchlosti.

Zdroje odometrie musia vysielať informácie pomocou knižnice `tf` o koordinačnom rámci, ktorý ho spravuje.

6.2.4 Ovládač základne

Navigačný zásobník predpokladá, že môže ovládať pohyb robota príkazmi o rýchlosti zasielaním `geometry_msg/Twist` správ s predpokladom, že sa nachádza v základnom koordinačnom rámci

s prístupom k topicu `cmd_vel`. To znamená, že musí existovať uzol, ktorý dokáže premeniť hodnoty rýchlosti (x,y,theta) na pohybové príkazy a poslať ich na pohybovú základňu.

6.3 Nastavenie navigačného zásobníka

Predpokladá, že všetky požiadavky uvedené vyššie sú splnené. To znamená, že každý senzor, ktorý budeme využívať bude vysielať informácie o koordinačnom rámci pomocou `tf`. Ďalej, vysielanie odometrických informácií pomocou `tf` a odometrických správ a prijímanie ovládania rýchlosť a jeho zasielanie na základňu pohybu.

Ďalej musíme vytvoriť package, ktorý bude obsahovať všetky súbory nastavení a spúšťacie súbory pre navigačný zásobník. Je dôležité vytvoriť balíček so závislosťami na každom balíčku použitom k nastaveniu robota ako aj na `move_base` balíčku tak aby bolo možné úspešne spustiť navigačný zásobník.

Po vytvorení pracovného priestoru pre naše konfiguračné a spúšťacie súbory potrebujeme vytvoriť súbor `roslaunch`, ktorý si vyvolá všetky potrebné hardwarové a transformačné publikácie. Ďalej potrebujem daný súbor nastaviť na nášho konkrétneho robota. Pre každý senzor, ktorý bude navigačný zásobník využívať. Treba vyvolať balíček s ROS ovládačom, typ ovládaču, názov uzla a ďalšie parametre potrebné pre spúšťaný uzol. Podobne je potreba spustiť aj odometický uzol a nakoniec aj transformačnú konfiguráciu.

6.3.1 Konfigurácia máp s oceneniami

Navigačný zásobník využíva dve mapy s oceneniami (ďalej costmap) na ukladanie informácií o prekážkach vo svete a to: globálne a lokálne. Globálnu mapu používa pre globálne plány (dlhodobé plány naprieč celým prostredím) a lokálnu mapu, ktorá slúži na vyhýbanie sa prekážkam v blízkom okolí. Máme nastavenia, ktoré chceme aby obidva typy máp dodržiavali a potom individuálne nastavenia pre každý typ zvlášť.

Obecne navigačný zásobník využíva costmapy na ukladanie informácií o prekážkach vo svete. Aby to mohlo fungovať potrebujeme nastaviť costmapy tak aby naslúchali aktualizáciám z tém senzorov. K tomu potrebujeme nastaviť prahy podľa ktorých budú senzory vyhodnocovať prekážky alebo voľný priestor. Ďalej potrebujeme nastaviť rozmery robota. Tie nastavujeme s predpokladom, že stred robota má hodnotu (0.0, 0.0). Zadávanie je možné v smere alebo v protismere hodinových ručičiek. Pre prípad kruhového tvaru robota sa zadá hodnota polomeru. Ešte potrebujeme zadať hodnotu inflačného polomeru, ktorý by mal byť nastavený na maximálnu vzdialenosť od prekážky v ktorej je ohodnotenie costmapy navýšené. Nakoniec potrebujem vytvoriť zoznam senzorov, ktoré budú dodávať informácie costmapám, a nastaviť parametre samotným senzorom zo zoznamu. Tieto parametre sú: názov koordinačného rámca senzora, typ správy podľa topicu, povolenie pridávať a odstraňovať informácie o prekážkach.

Pre nakonfigurovanie globálnej costmapy potrebujeme definovať rámec v ktorom bude spustená a rámec, ktorý bude odkazovať na základňu robota. Potom už len obnovovacia frekvencia v Hz.

Konfigurovanie lokálnej costmapy prebieha podobne ako pri globálnej, sú tam ale tieto parametre navyše. Parameter určujúci frekvenciu vysielania vizualizačnej informácie. Nastavenie centrovania mapy na robota a výšku, šírku a rozlíšenie costmapy.

6.3.2 Konfigurácia základného lokálneho plánovača

Lokálny plánovač je zodpovedný za prepočítavanie príkazov o rýchlosti a ich zasielanie do základne pohybu. Potrebujeme upraviť nastavenia podľa špecifikácií pre konkrétneho robota.

Lokálny plánovač poskytuje ovládanie, ktorým riadi základňu pohybu v rovine. Toto ovládanie slúži ako prepoj medzi trasovacím plánovačom a robotom. S použitím mapy, plánovač vytvára kinematickú trajektóriu pre robota od štartu k cieli. Počas cesty, plánovač ohodnocuje okolie robota pomocou funkcie ohodnocovania kde mapa je reprezentovaná mriežkou. Funkcia ohodnocuje prechody medzi bunkami mriežky. Úlohou ovládača je využívať tieto informácie a vyhodnotiť rýchlosti dx, dy ,dtheta a posielať ich robotovi.

6.3.3 Spúšťací súbor

Keď už máme všetky potrebné nastavenia a konfigurácie hotové, potrebujeme všetko spojiť dohromady do spúšťacieho súboru pre navigačný zásobník.

7 Návrh implementácie

V tejto kapitole sa pokúsim čo najpodrobnejšie popísať postup implementácie systému pre fúziu dát zo senzorov, pre ktorý som si vybral rozšírený Kalmanov filter. Kód programu som písal v jazyku C++.

Z môjho výskumu som sa dozvedel o dôležitých krokoch ktoré je potrebné spraviť aby robot bol schopný správne využívať informácie poskytované senzormi a vyhodnotiť podľa nich stav svoj a svojho okolia. Kroky potrebné na toto vyhodnotenie sú:

- **Robot:** Či už model robota alebo reálny, potrebujem zistiť rozmery robota a rozmiestnenie rôznych prvkov, ktoré robot používa.
- **Transformácia:** Pre dodanie kontextu senzorovým informáciám potrebujeme nastaviť kde a aké senzory sa nachádzajú.
- **Vysielanie a prijímanie informácií zo senzorov:** Nezávislý robot po nameraní odošle dané informácie korektnou správou skrz ROS (ako publisher). Správa je zaslaná na tému (topic), ktorá je pre daný senzor vytvorená. Z témy správu potom získa môj program (ako subscriber).
- **Fúzia:** Pre spracovanie informácií z viacerých senzorov uložím postupne informácie o meraniach do prvej matice. Do druhej matice vložím vzorce, ktoré ovplyvňujú stav robota. Pomocou týchto vzorcov získam informácie o zmene stavu robota.
- **Kalmanov filter:** je hlavným krokom tejto práce. Spracováva informácie zo senzorov a vyhodnocuje odhad stavu robota. Pracuje v dvoch fázach, predpoveď a korekcia. Avšak prechádzajúci krok, fúzia, je prakticky už súčasťou filtra.
- **Lokalizácia:** Z Kalmanovho filtra dostaneme odhad stavu robota po každej jeho iterácií. Odhad využíva lokalizačný systém k zobrazeniu pozície robota na mape.

Pre testovanie filtra som používal simulačné prostredie Gazebo v ktorom som si vytvoril model robota s rôznymi senzormi a RViz v ktorom som porovnával reálny stav robota a porovnával so stavom vypočítaným cez Kalmanov filter.

7.1 Vytvorenie modelu robota

Z dôvodu testovania potrebujem vytvoriť jednoduchý model robota do simulácie. Model bude obsahovať senzory, ktoré budú vysielať simulované dáta pre kontrolu a testovanie lokalizačného programu. Prvky modelu robota je potreba nastaviť tak aby boli schopné čo najvernejšie simulovať realitu.

7.1.1 Základné nastavenie

Aby sme mohli začať, musíme zhodnotiť koľko a akých koordinačných rámcov budeme potrebovať. Každý senzor by mal mať vlastný rámec. Ďalej potrebujeme rámec pre odometriu a samozrejme rámec základne. Keď už máme potrebné rámce potrebujeme zistiť vzťahy medzi rámcami. Tieto vzťahy zodpovedajú rozdielu polohy a rotácie medzi rámcami v osách x, y a z (v metroch) pre polohu a sklon, natočenie a výchylka (v radiánoch) pre rotáciu. Vzťahy sa skladajú z rodiča a potomka kde hociktorý potomok môže mat práve jedného rodiča. Zo vzťahov vytvorím transformácie a z nich následne vznikne transformačný strom. Aby mohol tento strom vzniknúť musia rámce svoje transformácie vysielať (publikovať) a zároveň musí existovať uzol ktorý týmto vysielaniam naslúcha.

7.1.2 Vytvorenie modelu a transformácií

Keď už máme náhľad aké potrebujeme rámce začneme vytvárať samotného robota. Model robota pre simulátor budeme písať v jazyku *SDF*, ktorý používa syntax *xml.*

Pre vytvorenie nového modelu si vytvoríme súbor s názvom robota *.config* a *.sdf*. Do súboru *.config* si nastavíme fyzické atribúty modelu a jeho počiatočnú pozíciu na mape. Tieto atribúty vpisujeme pod značky *visual*. Aby však model dokázal aj interagovať s okolím musíme nastaviť aj atribúty v *collision.* Collision by mal byť len zjednodušenou verziou vizualu aby zbytočne nezaberal výpočet a modelovanie kolizných parametrov výpočetnú silu simulátora.

Preto pre nášho robota zvolíme „bublinu“ okolo robota ako kolizný atribút. Ako vizuál zvolíme jednoduchý hranol ako chassis s dvomi oválnymi kolečkami, na každej strane jedno. Kolečká pripojíme k chassis pomocou *joint* značky. Ešte pridáme nejaké senzory a máme robota na simuláciu.

Nato aby mohol byť model úspešne premietnutý na mape simulátora musíme ešte do súboru *.sdf* priradiť transformácie robota na svet. Taktiež nastavíme vysielanie odometrie na požadovaný topic (*/odom*). Senzory, ktoré sme si pridali na modle je potrebné tak isto nastaviť aby vysielali svoje správy na topicy. K správnemu fungovaniu ale senzory potrebujú ovládač. Ten priradíme k senzoru pod značkou *plugin* ako aj topic na ktorý bude senzor vysialať.

Nutnou značkou každého modelu v simulátore je *inertia*, ktorá určuje fyziku modelu.

7.2 Práca s ROS-om

Každý senzor odosiela správy s dátami. Správy odosielajú senzory na témy kde každý senzor posiela informácie a vlastnú tému. Z tém získavam dáta subscriberom na dané témy. Dáta získané zo senzorov ukladám do matice filtra tak aby boli spracované správnym spôsobom.

Subscriber používa callback funkciu, pomocou ktorej získava vždy najakuálnejšie dáta z témy. Preto aby subscriber fungoval ako má musia callbackové funkcie byť v nekonečnej cykle. Na každú tému (resp. senzor) potrebujem vytvoriť samostatný subscriber.

Správy posielané na tieto témy obsahujú časovú pečiatku aby bolo možné vyhodnotiť rozdiel medzi aktuálnym a predchádzajúcim stavom ak by sa správy z nejakého dôvodu oneskorili.

Podobne ako každý senzor potrebuje vlastnú tému tak každý druh senzoru potrebuje správnu štruktúru správy. To znamená, že pre každý typ správy potrebujem vytvoriť aj jej správne spracovanie. Informácie zo senzorov sú spracovávané v callbackových funkciách.

7.3 Rozšírený Kalmanov filter

Pre správne fungovanie filtra potrebujem zaviesť matice do kódu. C++ v základe nepodporuje matice a tak som si zvolil knižnicu Eigen, ktorá poskytuje dostatočné operácie nad maticami pre moje potreby.

Matice potrebujú správne rozmery nato aby pracovali tak ako majú. Napríklad výsledná stavová matica bude obsahovať všetky stavové premenné, ktoré si užívateľ zvolí aby filter vyhodnotil. Niektoré z matíc sú konštantné ale aj tak musia mať veľkosť podľa nastavených parametrov ako počet vstupov a výstupov.

Keď už máme nastavené matice tak ako majú byť potrebujeme naplniť vstupné matice dátami z meraní. To sa vykonáva už v callbackovej funkcií.

Po príchode dát z merania sa spustí iterácia filtra. Ten zo získaných informácií vypočíta odhadovaný stav robota v priestore. Túto informáciu potom odošle na tému pre ďalšie použitie. V mojom prípade testovanie v programe RViz.

7.3.1 Vytvorenie základu

Ako základ som si teda vytvoril triedu *kalman_filter*. Pri inicializácií triedy sa nastavia hodnoty filtra tak aby matice neporušovali pravidlá operácií nad maticami.

Fúziu meraní vykonáme tak, že pri príchode informácií zo senzoru prevedieme získané hodnoty na jednotky s ktorými filter vie počítať a do súradného systému v ktorom sa nachádza stav robota. Prevedené hodnoty potom uložíme do matice pomocou ktorej bude filter prijímať nové merania a upravovať nimi stav robota.

Pre správnu fúziu meraní v Kalmanovom filtri potrebujeme ešte nastaviť maticu, ktorá bude hovoriť akým spôsobom má ten alebo ten senzor ovplyvňovať stavové hodnoty. V matici sa môže nachádzať jednotka pre priame ovplyvnenie hodnoty ako aj určitý vzorec.

Pri prvom priechode cyklu filtra si nastavíme hodnotu stavového vektoru podľa nameraných hodnôt zo senzorov. Tým zabezpečím aby hodnoty získané po vyhodnotení celého cyklu boli korektné. Následne a v každej ďalšej iterácií sa volá funkcia predpovede *prediction*. Funkcia obsahuje rovnice 5.4.17 a 5.4.18 pomocou ktorých sa vypočíta „priori“ stavový vektor a procesová kovariancia.

Po výpočte „priori“ hodnôt sa zavolá funkcia *correction* ktorá má opraviť predpoveď predchádzajúcich výpočtov. Pre výpočet optimálneho Kalmanovho zisku sa vypočíta rovnica 5.4.19. Takto vypočítaný zisk nám pomôže pri vyhodnotení dôveryhodnosti meraní v rovnici 5.4.20 a korekcií „posteriori“ stavového vektoru ktorý je výsledkom jak tejto rovnice tak aj filtra. Nakoniec ešte opravíme kovarianciu a celý cyklus môže začať odznova.

7.3.2 Generalizácia

Keď nám už funguje základ filtra môžeme ho začať rozširovať a generalizovať aby dokázal prijať čo najväčšie množstvo rôznych správ od rôznych senzorov. Pod generalizáciu rozumiem úpravu vstupu a nastavenia filtra tak aby bolo možne jeho použitie na ľubovoľného robota s ľubovoľnými senzormi a bez zásahu do samotného kódu filtra. Ďalej popisujem vstupy ktoré pri správnej forme dokáže filter spracovať.

7.3.2.1 Pozícia

Vo funkcii *setPosition_estimation* je možné nastaviť ktoré hodnoty bude filter využívať na určenie polohy. Môžu to byť pozícia v x, y, z osiach, rýchlosť týchto osiach a akcelerácia. Funkcia nastavuje stavovo-tranzitívnu maticu tak aby hodnoty správne ovplyvňovali výstup z filtra. V základe je to identitná matica rozšírená o rovnice pre vyhodnotenie pozície z rýchlosti alebo akcelerácie.

Táto funkcia dokáže prijímať a spracovávať aj hodnoty vzdialenosti a náklonu, na základe ktorých je možné výslednú pozíciu vypočítať. Tento spôsob vyhodnocovania je z podľa mňa dosť zastúpený, aj keď sa nejedná o štandardný postup, a tak si vyslúžil miesto v mojom programe.

Ďalšia funkcia, ktorá sa týka pozície, je jej výpis na výstup resp. či bude filter vyhodnocovať stav robota v určitých osách. Je to funkcia *setPosition_output* a mení maticu meraní tak aby zodpovedala výstupu, ktorý požadujeme.

7.3.2.2 Orientácia

Podobne ako pre pozíciu aj pre orientáciu sú funkcie, ktoré nastavujú filtrovanie s hodnotami orientácie.

Tu je vzniká problém s výpočtami. Je to z dôvodu že orientácia sa počíta buď pomocou Eulerových uhlov alebo Quaternionov. Keďže ale správy v ROS-e používajú prevažne quaterniony na vyjadrenie orientácie v trojrozmernom priestore, primárne aj môj filter počíta s nimi. Zároveň ale dokáže pracovať aj s Eulerovými uhlami.

7.3.2.3 GPS

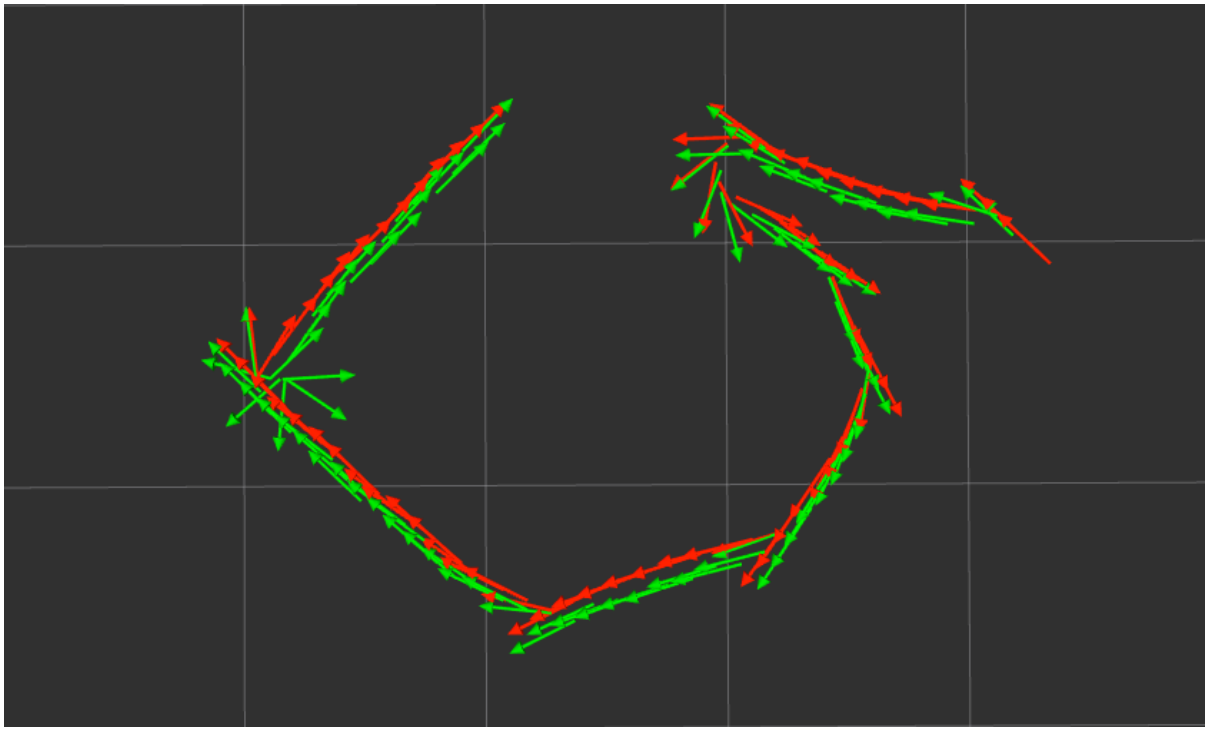
Dáta prijímané z GPS je náročnejšie na spracovanie. Keďže GPS posiela zemepisnú dĺžku a šírku, potrebujeme tieto merania previesť na rámec mapy v ktorej sa robot pohybuje. Po transformácií na rámec našej mapy nám GPS poskytuje hodnoty, ktoré môžeme priamo použiť vo filtri až na orientáciu v theta,.

7.4 Presnosť lokalizácie

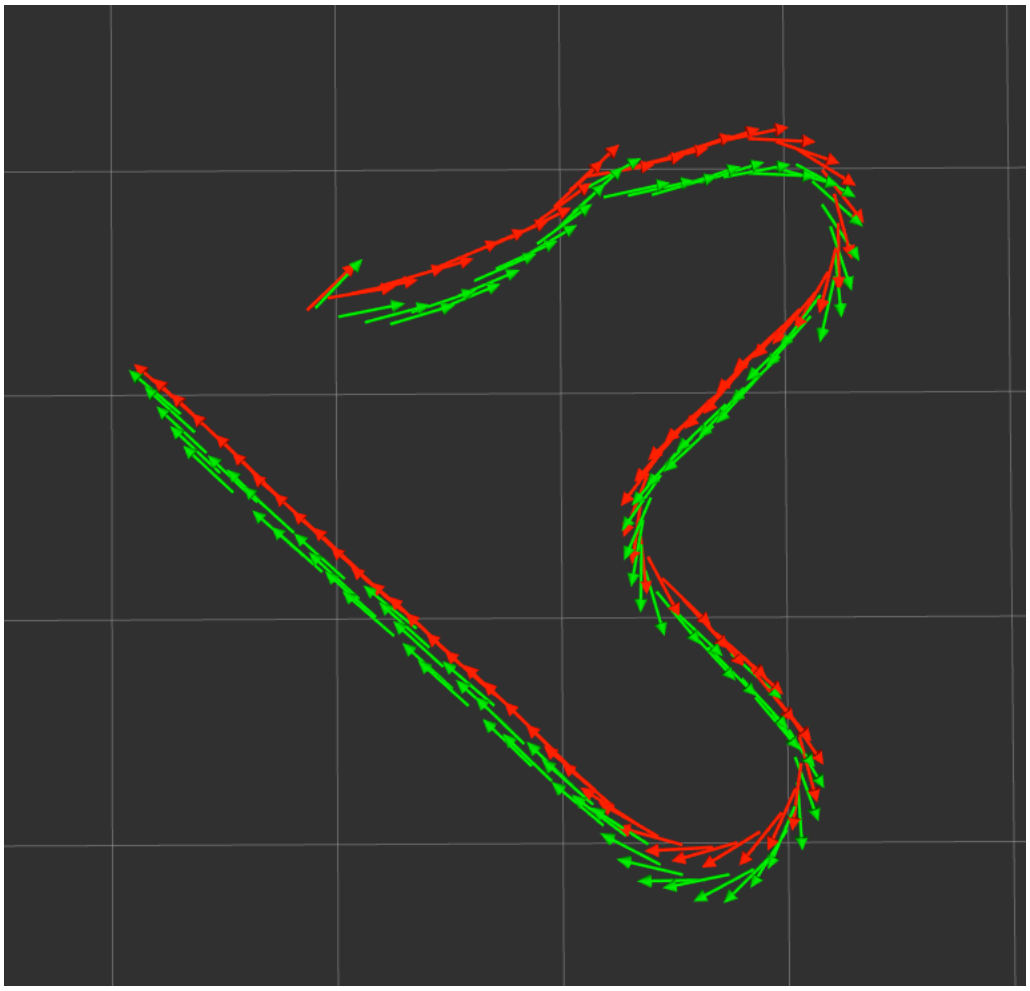
Pre otestovanie presnosti lokalizácie som použil program RViz, ktorý prijíma správy a umožňuje ich vykreslovanie na mapu. Pre naše testovanie sme použili zobrazenie odometrie, pretože zobrazuje jak pozíciu a orientáciu tak aj predchádzajúce stavy čo nám umožňuje porovnať presnosť vyhodnotenia Kalmanovým filtrom oproti reálnej pozícií robota. Robot a jeho dáta sú simulované pomocou simulátora Gazebo.

Na Obrázku 7.1 môžeme vidieť že vyhodnotená pozícia (zelená) má malé odchýlky od reálneho stavu (červená). Na tomto obrázku sme testovali hodnoty primárne na priamom pohybe a ostrých zmenách smeru. Až na posledné 90° otočenie, pri ktorom na krátku dobu výrazne chyboval v orientácií, je vyhodnotená pozícia velice presná. Pri druhej simulácií sme zobrazenej na Obrázku 7.2 sme vyskúšali ako si poradí s postupnejšou zmenou smeru počas pohybu vpred. Ani tu si filter neviedol zle. Nakoniec sme otestovali ako lokalizácia zvládne násilný presun robota do inej polohy s inou orientáciou. Na Obrázku 7.3 vidíme odozvu filtru počas presunu robota (časť viacerých zelených šípok bez červených v okolí). Podľa toho usudzujem, že simulátor odosielal dáta aj počas tohto časového úseku a filter sa tak bez problémov zorientoval.

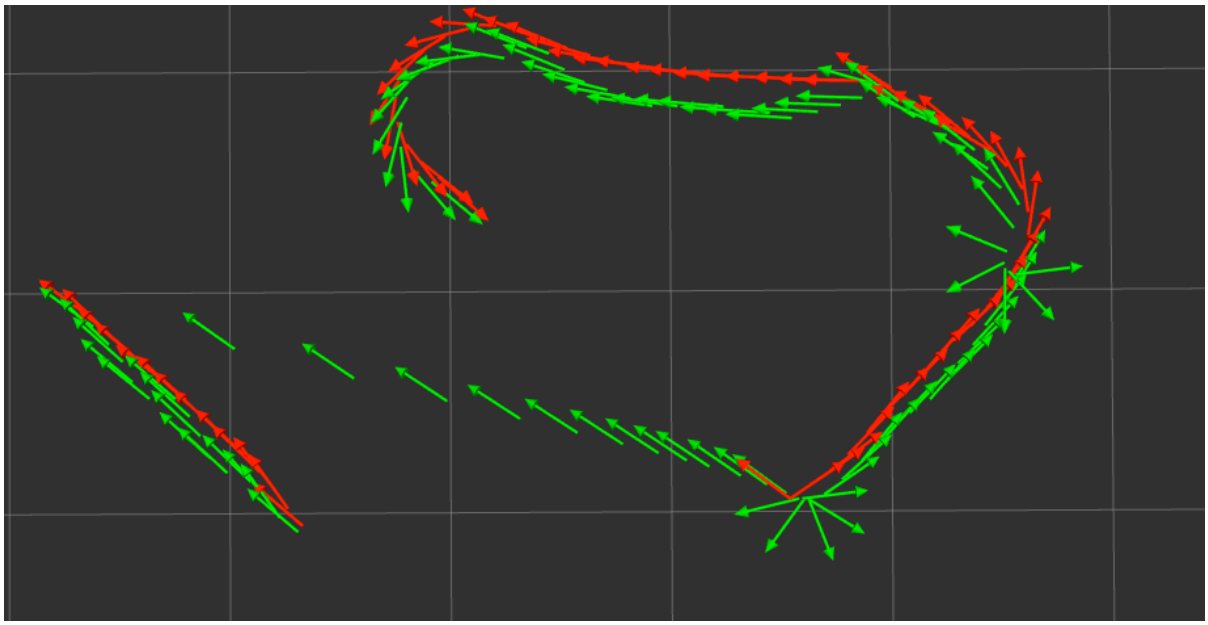
Z testovania v simulovanom prostredí môžeme zhodnotiť, že presnosť vyhodnotených údajov je dostatočne presná aby sa podľa nej robot orientoval. Hodnoty získané z filtra až na niekoľko výnimočných stavov líšia len o malú časť. Keďže ale simulátor neberie v úvahu šum vznikajúci pri meraniach, ďalšie testovanie na reálnom robotovi by bolo viac než vhodné.



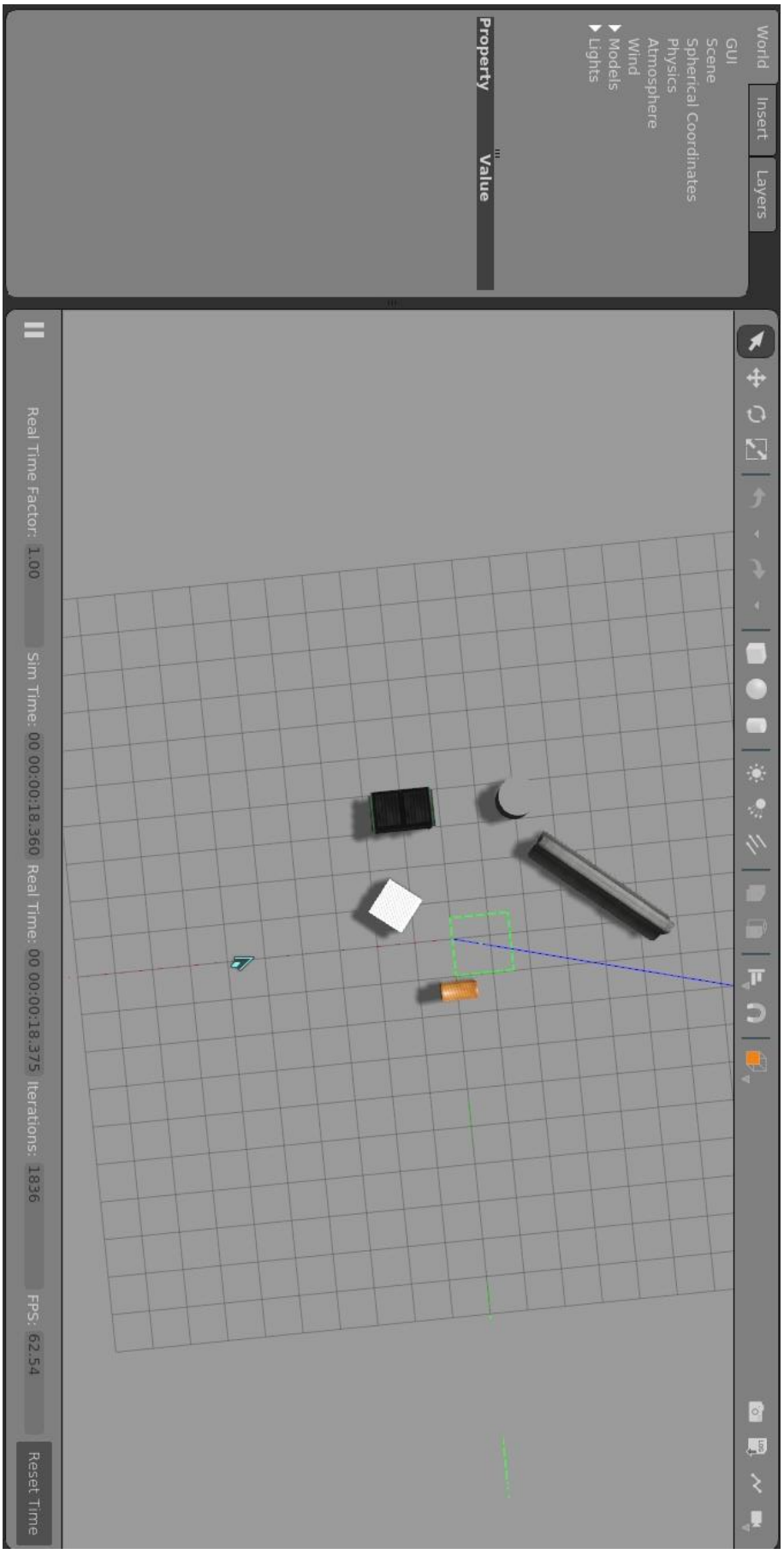
Obrázok 7.1 - Testovanie presnosti pri priamom pohybe



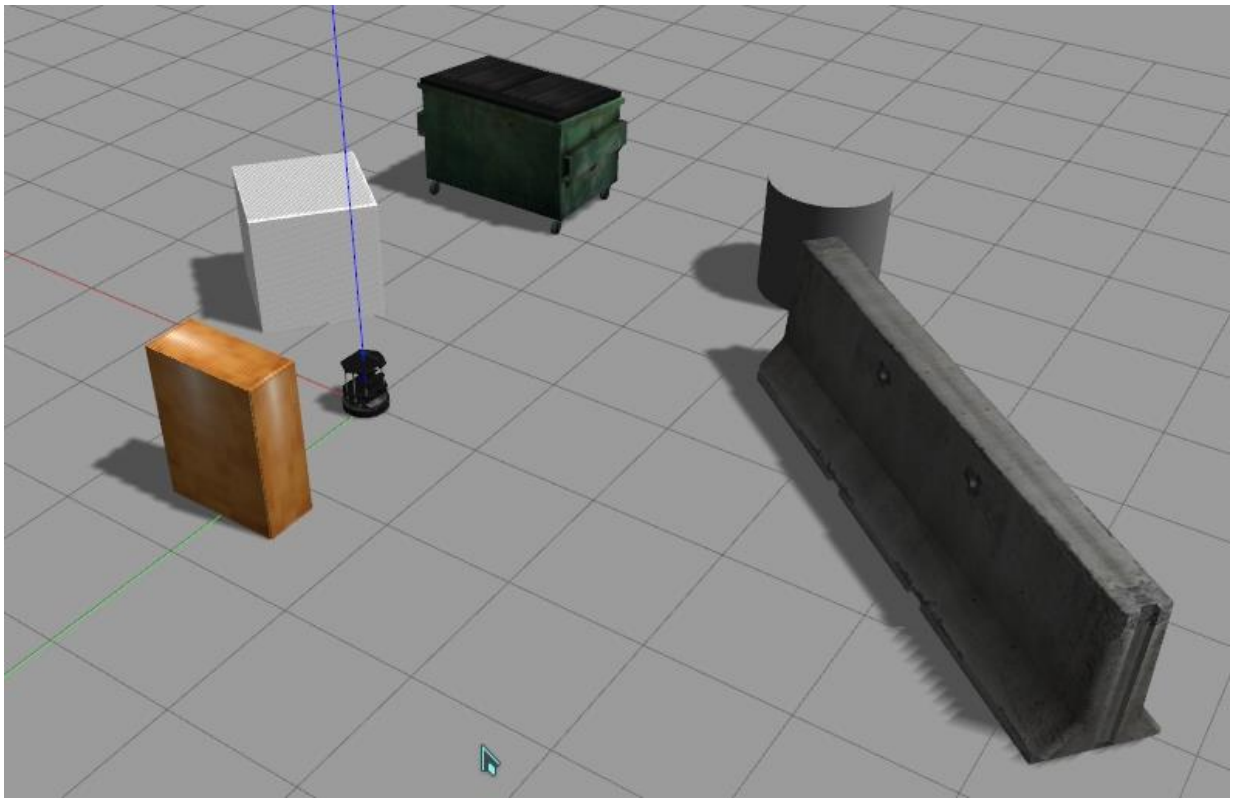
Obrázok 7.2 – Testovanie presnosti pri zmene smeru počas pohybu



Obrázok 7.3 – Testovanie správania filtra pri dislokácií robota



Obrázok 7.4 – Testovacie prostredie Gazebo



Obrázok 7.5 – Testovacie prostredie Gazebo s modelom robota

8 Závěr

Problematika lokalizácie robota je v robotike jedna z najdôležitejších odvetví, pretože každý pohybujúci sa robot potrebuje nejakým spôsobom vedieť kde sa nachádza. Pravdepodobnostná robotika je jedným zo spôsobov ako pristupovať k tomuto problému. Na rozdiel od ostatných lokalizačných metód nezakladá na jedinom najlepšom odhade ale na pravdepodobnostnom rozložení, ktoré obsahuje možné hypotézy ponad celým priestorom.

Poloha robota sa určuje podľa meraní zo senzorov a aj z riadenia robota. Prenos informácií zabezpečuje ROS cez témy. Informácie zo senzorov a riadenia spracujeme dohromady pre lepšiu manipuláciu. Tento proces sa nazýva fúzia senzorov.

Fúziu potom dostane filter. Rozšírený Kalmanov filter, ktorý som si vybral, vyhodnocuje odhad polohy robota v priestore v reálnom čase. Pracuje v dvoch krokoch, rekurzívne. Jeho najdôležitejšou výhodou je nutnosť pamätať si len predchádzajúci stav robota a jeho rýchlosť vyhodnocovania. Počet senzorov použitých ovplyvňuje presnosť odhadu a dĺžku výpočtu.

Moja adaptácia filtra umožňuje úpravu vstupu ako aj výstupu. Na vstupe je možné nastaviť aké senzory budú informácie dodávať a aké hodnoty budú dané senzory poskytovať. Na výstupe zase umožňuje výber stavových hodnôt ktoré filter bude odosielať na tému.

Pri testovaní som zistil, že je možné určovať polohu robota prakticky len za pomoci rozšíreného Kalmanovho filtra. Jeho odhad je väčšinou dostatočne presný nato aby robot sa zvládol podľa neho orientovať v priestore. Na druhú stranu bol filter testovaný len na simulácií a za ideálnych podmienok. Preto predpokladám, že pri nasadení na reálneho robota by sa ukázaly nedostatky, ktoré sa v simulovanom prostredí len ťažko testujú.

Pre ďalší vývoj by pomohlo skúsiť filter na reálnom robotovi s laboratórnymi podmienkami. Takýto test by ukázal problémy implementácie na konkrétneho robota zatiaľ čo ideálne podmienky by tento proces urýchlili. Prístupnosť filtra by zabezpečilo rozšírenie kompatibility prijímaných správ a podpory väčšej škály senzorov. Pretože namerané hodnoty senzorov majú rôznorodú štruktúru je prakticky nemožne sa prispôsobiť všetkým. Je ale možné pokračovať v generalizácií programu a tak zaistiť čo najmenšie úsilie potrebné k nastaveniu filtra pre začatie jeho používania.

Literatúra

[1] THRUN, Sebastian, Wolfram BURGARD a Dieter FOX. *Probabilistic robotics*. London, England: MIT Press, 2006. ISBN 978-0-262-20162-9

[2] BUCKL, Katharina. *Sensor Fusion using the Kalman Filter* [online]. 8.3.2005. [Cit. 14.5.2017]. Dostupné z: http://campar.in.tum.de/Chair/KalmanFilter

[3] LEVY, Simon D. *The Extended Kalman Filter: An Interactive Tutorial* [online]. 19.12.2016. [Cit. 14.5.2017]. Dostupné z: https://home.wlu.edu/~levys/kalman_tutorial/

[4] Články. *Bzarg.com: How a Kalman filter works, in pictures* [online]. 11.8.2015. [Cit. 14.5.2017]. Dostupné z: http://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/

[5] RISTIC, Branko., Sanjeev. ARULAMPALAM a Neil GORDON. *Beyond the Kalman filter: particle filters for tracking applications*. Boston, MA: Artech House, 2004. ISBN 158053631X.

[6] WELCH, Greg a Gary BISHOP. *An Introduction to the Kalman Filter* [online]. Posledná úprava 24.6.2006. [Cit. 14.5.2017]. Chapel hill: Department of Computer Science University of North Carolina at Chapel Hill. Dostupné z: http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf

[7] ZACHAIRE, Mbaitiga. *GPS and Discrete Kalman Filter for Indoor Robot Navigation* [online]. 2011. [Cit. 14.5.2017]. Dostupné z: http://web.cecs.pdx.edu/~mperkows/temp/KALMAN/KALMAN-PAPERS/GPS=Kalman-Good.pdf

[8] *Robot operating system (ROS).* ISBN 9783319549262.

[9] QUIGLEY, Morgan, Brian GERKEY a William D. SMART. Programming robots with ROS. ISBN 1449323898.

[10] MARTINEZ, A. a FERNÁNDEZ E.. *Learning ROS for Robotics Programming*. Pact Publishing. 25.9.2013. ISBN 9781782161455.

[11] *Documentation* – ROS Wiki [Online; Cit. 18.3.2018]. URL: http://wiki.ros.org/

[12] *Definition of C++ Programming Language, LISP, Python* – techopedia [Online; Cit. 6.4.2018]. URL: https://www.techopedia.com

[13] SKALKA, Marek. *Srovnání lokalizačních technik v robotice* : Diplomová práca. Praha : Univerzita Karlova v Praze. 2011.

[14] THRUN, Sebastian. *Probabilistic Algorithms in Robotics* [online]. 2000. Dostupné z: https://pdfs.semanticscholar.org/d20a/826a9d0ff85336f1c52d65dc06ed4c642703.pdf

[15] ELMENREICH, Wilfried. An *Introduction to Sensor Fusion* [online]. 19.11.2002. Vienna University of technology, Austria. Dostupné z: https://www.researchgate.net/profile/Wilfried_Elmenreich/publication/267771481_An_Introduction_to_Sensor_Fusion/links/55d2e45908ae0a3417222dd9/An-Introduction-to-Sensor-Fusion.pdf

Seznam příloh